METHODS FOR COLLABORATIVE CONCEPTUAL DESIGN OF AIRCRAFT POWER ARCHITECTURES

A Dissertation
Presented to
The Academic Faculty

by

Cyril de Tenorio

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Aerospace Engineering

Georgia Institute of Technology
AUGUST 2010

METHODS FOR COLLABORATIVE CONCEPTUAL DESIGN OF AIRCRAFT POWER ARCHITECTURES

Approved by:

Dr. Dimitri Mavris, Advisor
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Elena Garcia
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Nicolas Antoine
System Architect – Future Progammes
Airbus S.A.S.

Dr. Christiaan Paredis
School of Mechanical Engineering
Georgia Institute of Technology

Dr. Brian German
School of Aerospace Engineering
Georgia Institute of Technology

Date Approved: May 2010

To Henri and Jeanne Meunier

Acknowledgements

I wish to aknowledge with gratitude my debt toward my adviser Professor Mavris. From the very first day, he has believed in my ability to do great things. His trust and recommendation have led me to extraordinary opportunities to learn and grow. I will always treasure the trust and advice he awarded me. I also would like to express my gratitude toward my dear supervisor and friend Dr. Garcia. Her patient encouragements and wise advice have been a blessing.

I would like to thank Dr. Antoine for his wise guidance and his ability to take a step back and observe the underlying mechanisms in the industry. For these reasons, he has been an inspiration for me. I also express my gratitude to Dr. German for his acute remarks and insights. I also would like to thank Prof. Paredis for his great influence on this thesis. His passion and great analytical skill in the field of design theory have been an important building block in the formulation of the ideas presented in this thesis.

This thesis was performed jointly between EADS, Airbus and the Georgia Institute of Technology (Georgia Tech). This context allowed this dissertation to benefit both from an outstanding academic context with a direct industrial exposure. This double context gave a special flavor to the ideas presented in this dissertation. For that opportunity, I present my grateful regards to Dr. Gardin from the EADS corporate technical office. Nothing would have been possible without his visionary leadership and trust. I also would like to thank Martine Callot and her team (Dr. Riviere and Dr. Reyterou) for providing me with the help and environement necessary to explore and understand the industrial context of this thesis.

I would like to thank my family: My brother Alexandre for being here for me, my Mother for her love, drive, help, and support. I also would like to aknowedge my dear Grand-Parents Jeanne “Nannie” and Henri Meunier. Along with my mother, they have played a major role in my education. From Nannie, I have inherited my exigence and

attention to details. From my Grand-Father, I have received my sense of responsibility and passion for engineering. Ever since my first Math problem and discordant note on my violin, they have been here for me. For that, and all the love and support they provided along the way, I would like to dedicate this thesis to them.

Finally, I like to thank my friends in Atlanta at Georgia Tech and the Aerospace System Design Lab. Some of my friends deserve more gratitude than I could ever express in these pages, my dear Olivia "Poulette" Pinon for her unconditional friendship, and affection, my brother Jason "Figaro" Sherwin for his flamboyant and positive character, my friend Robert "Chou" Combier for his extraordinary creativity, talent and humor.

Only the harmony of friendship

Makes milder the burdens,

Without this sympathy

There is no happiness on earth.

Pamina & Papageno

Die Zauberflöte - Wolfgang Amadeus Mozart and Emanuel Schikaneder

But among all my friends, I would like to thank my dearest friend of all: Joanne, my wife and *Pamina*. Through these years of obstinated efforts, she has been my "brightest light in my darkest night".

Table of Contents

Acknowledgements ..... **iv**

List of Tables ..... **xix**

List of Figures ..... **xxiii**

Summary ..... **xxxv**

Chapter 1 Motivation **1**

1.1 General Introduction 1

1.2 Aircraft System and Architectures 2

1.3 Evolutions of Aircraft Functionalities 3

1.4 Evolution of Aircraft System Technologies 5

1.5 Development of Complex Systems within a Multi-Organizational Environment 6

1.6 Conclusion 9

Chapter 2 Problem Definition **11**

2.1 Complex System Design and its Development Cycle 12

2.1.1 Defining “design” 12

2.1.2 Development Cycle Implied by Design 15

2.1.3 Concepts of Concurrent Engineering 17

2.1.4 Implementation Approaches to Concurrent Engineering 23

2.2 Design in an Industrial Context 25

2.2.1 Challenges due to the Project Scale 25

2.2.2 A culture of clustered-concurrent design 26

2.2.2.1 Definition of Clustered Concurrent Engineering Developments 26
2.2.2.2 The Essential Role of Clustered Concurrent Engineering 28
2.2.2.3 Clustered Concurrent Engineering Role in Architectural Conservatism .. 28
2.3 Challenges in Applying Concurrent Engineering Approaches to Architecture Conceptual Development 30
2.3.1 Overview of Conceptual Activities 30
2.3.2 The Role of the Architect 33
2.3.3 Integration of Knowledge 35
2.3.3.1 Communication between Experts: 37
2.3.3.2 Promptness of Execution 38
2.3.3.3 Complexity of the Analysis 38
2.3.3.4 Estimation of Subsystem Optimization Potential in Architectural Sizing 41
2.3.3.5 Tracing Assumptions 43
2.3.3.6 Human Factors 46
2.3.3.7 Conclusions 47
2.3.4 Deduction from Previous Descriptions 48
2.4 Linking Conceptual Design to Clustered Concurrent Preliminary and Detailed Design 49
2.4.1 Theoretical Description of a Complex Design Process 50
2.4.2 Architectural Design Role with Respect to Clustered Concurrent Developments 51
2.4.2.1 Formalization of Subsystem Design 51
2.4.2.2 Objectives of the Architecture Design Problem 52
2.4.2.3 Formalization of the Architectural Design Problem 53
2.4.3 Consequences of Conceptual Design Approaches on the Detailed Design Problem Formulation 55

2.4.3.1 Description of the Traditional Conceptual Design Approach 55
2.4.3.2 Difficulties Implied by Traditional Approaches in a Subsystem Design Context 57
2.4.3.3 Subsystem Intrusive Conceptual Design Approaches 58
2.4.4 Problem Associated with Traditional Conceptual Methods in an Architectural Context 59
2.4.4.1 Distinction between Targets and Objectives 59
2.4.4.2 Illustration of Improper Formulation of Subsystem Design Problems 62
2.4.5 Deduction from Previous Observations 65
2.5 Research Objectives and Research Questions 66
Chapter 3 State of the Art Review and Observations 70
3.1 Methods for Requirement Formulation 71
3.1.1 Functional Analysis 72
3.1.1.1 Functional Tree 73
3.1.1.2 Classification of Functions 74
3.1.1.3 Function Analysis System Technique (FAST) 75
3.1.1.4 Quantitative Use of Functions 77
3.1.1.5 Synthesis of Functional Analysis Techniques 77
3.1.2 Objective Analysis Methods 78
3.1.2.1 Definition of Objectives 79
3.1.2.2 Objective Tree 79
3.1.3 Overview of Requirement Formulation Methods 81
3.2 Methods for Architecture Concept Definition 83
3.2.1 Decomposition and Composition Methods 84
3.2.1.1 Decomposition Methods 84
3.2.1.2 Matrix-based Composition Methods 92

3.2.1.3 Language-based Composition Methods 95
3.2.2 Architecture Structural Definition Methods 97
3.2.2.1 Interaction Graphs and Matrices 98
3.2.2.2 Genetic Network Strings 100
3.2.2.3 Descriptive Languages 103
3.2.3 Comparison of Architecture Definition Methods 104
3.2.3.1 Selection of Criteria for Comparing Definition Methods 104
3.2.3.2 Test Case Application for the Experimentation 105
3.2.3.3 Deployment of Architecture Definition Methods 107
3.2.3.4 Conclusions about Architecture Conceptual Definition 112
3.3 Modeling and Analysis Approaches 116
3.3.1 Expert knowledge integration approaches 116
3.3.1.1 Quality Function Deployment 116
3.3.1.2 Multi-Disciplinary Analysis 120
3.3.1.3 Discussion on Modeling Approaches 128
3.3.2 Analysis Approaches 129
3.3.2.1 Bottom-up Analysis 130
3.3.2.2 Derivative Analysis 130
3.3.2.3 Review of Analysis Approaches 132
3.4 Review and Example of Conceptual Architecting Processes 134
3.4.1 Technology Mapping Methodology Roadmap 135
3.4.2 Quantified Relationship Matrices for Fuel Systems 136
3.4.3 Strategic Technology Planning 139
3.4.4 A Simulation Framework for Aircraft Power Management 141
3.5 Conclusions 143
Chapter 4 Proposed Methodology 144

4.1 Introduction 144
4.1.1 Proposed Architecting Framework 144
4.1.2 State of the Art Gaps in the Deployment of this Approach 146
4.1.2.1 Formulating the Mission 147
4.1.2.2 Automated Setup of the Model 147
4.1.2.3 Encapsulating Subsystem Knowledge 148
4.1.2.4 Setting up Subsystem Design Problem 148
4.1.3 Overview of Proposed Process 149
4.2 Architecture Requirement Definition 152
4.2.1 Identification of Boundary Functions and Objectives 153
4.2.1.1 Functional Decomposition 153
4.2.1.2 Definition of Architecture Level Objectives 154
4.2.1.3 SysML Representation 154
4.2.2 Mission Parameterization 155
4.2.2.1 Flight Phase 156
4.2.2.2 Flight Conditions 157
4.2.2.3 Criticality level of the Failure Configuration 157
4.2.3 Requirement Quantification 160
4.2.3.1 Overview of the Method for Scenario and Functional Profile Definition 161
4.2.3.2 Modeling Scenarios in SysML 162
4.2.4 Conclusion 163
4.3 Subsystem Knowledge Preparation 164
4.3.1 Identification and Classification of Subsystem Candidates 165
4.3.1.1 Identification of Subsystem Types 165
4.3.1.2 Functional Analysis of Subsystem Types 168

4.3.1.3 Verifying Completeness of the Subsystems Library 170
4.3.2 Negotiation of Modeling Interfaces 170
4.3.2.1 Definition of Functional Flows 171
4.3.2.2 Modeling Interfaces of Subsystem Sizing Models 175
4.3.2.3 Conclusions about Modeling Interfaces 178
4.3.3 Packaging Design Knowledge for Subsystem Sizing 179
4.3.3.1 What is Sizing? 179
4.3.3.2 Different Forms of Approximation: 181
4.3.3.3 Discussion 185
4.3.4 Conclusion 186
4.4 Sizing and Optimization Approach 187
4.4.1 Theoretical Perspective on Architecture Design Activities 188
4.4.2 Coordinated Optimization or Architecture-Level Steering of Subsystem Optimization 190
4.4.2.1 Optimization Process at Subsystem-Level 190
4.4.2.2 Optimization at System-Level 193
4.4.3 Interpretation and Implication of Coordinated Optimization 196
4.4.4 Conclusion on the Sizing Approach 197
4.5 Architecture Concept Definition 198
4.5.1 Composition: Declaration of Subsystems 202
4.5.1.1 Creation of the Architecture and Identification of Subsystem Types 202
4.5.1.2 Specification of Number of Instances and Names of Subsystems 203
4.5.2 Structure: Formulation of the Architecture Configurations 203
4.5.2.1 Listing Architecture Configurations 204
4.5.2.2 Definition of Architecture 205
4.5.3 Operation: Definition of Sizing Scenarios and Sequences 207

4.5.3.1 Definition and Scheduling Scenarios 207
4.5.3.2 Characterization of Functional Requirements 209
4.5.3.3 Assigning Architecture Configurations to Scenarios 210
4.5.4 Conclusions 211
4.6 Automated Model Construction 212
4.6.1 Phase 1: Importing the Architecture Composition 216
4.6.1.1 Description 216
4.6.1.2 Example 216
4.6.2 Phase 2: Importing the Mission 219
4.6.2.1 Description 219
4.6.2.2 Example 219
4.6.3 Phase 5: Building the Structure 223
4.6.3.1 Description 224
4.6.3.2 Example 225
4.6.4 Conclusions 227
4.7 Transition to Experimentation 227
Chapter 5 Verification of Hypotheses 229
5.1 Experiment 1: Characterization of a Commercial Aircraft Mission 232
5.1.1 Application of the Methodology 233
5.1.1.1 Functional Breakdown 233
5.1.1.2 Definition of Flight Phases 236
5.1.1.3 Definition of criticality levels 241
5.1.1.4 Elicitation of Requirements 245
5.1.2 Discussion of Results 246
5.1.2.1 Temporal Decomposition 247
5.1.2.2 Functional Decomposition 247

5.1.2.3 Criticality Level Classification 248
5.1.2.4 Conclusions of Experiment 1 249
5.2 Experiment 2: Sizing of subsystems 250
5.2.1 Deployment of the Optimizer-Based Sizing Methodology 252
5.2.1.1 Creation of the Optimizer-Based Sizing Model 252
5.2.1.2 Addressing Challenges 253
5.2.2 Presentation of a Functional Regression Sizing Model 257
5.2.3 Comparison of Results and Conclusions about Sizing Methods 257
5.2.3.1 Ability to Capture Subsystem-Level Trade-offs 257
5.2.3.2 Practicality for the Development of Sizing Models 262
5.3 Experiment 3: Coordinated Optimization of Subsystems 264
5.3.1 Overview of the experiments 264
5.3.2 Experiment 3.1: Formal Analysis of the Coordinated Optimization 265
5.3.2.1 Formal representation of the multi-level optimization problem 265
5.3.2.2 Taylor Series Expansion of the System-Level Objective 267
5.3.2.3 Applying the Condition to a Weighted Sum Subsystem-Level Objective 268
5.3.2.4 Validation of Hypothesis and Discussion 268
5.3.3 Description of the Architecture Test-Case used in Experiments 3.2 and 3.3. 269
5.3.3.1 Description of the Simple Architecture 271
5.3.3.2 Introduction of experiment 3.2 and 3.3 274
5.3.4 Experiment 3.2: Qualitative Analysis of Coordinated Optimization 274
5.3.4.1 Overview of the Test 274
5.3.4.2 Observation of Topologies Implied by the Optimization Processes 280
5.3.4.3 Conclusions 289
5.3.5 Experiment 3.3: Comparison of Subsystem Design Problem Formulation ... 290

5.3.5.1 Presentation of the test considered by experiment 3.3 291
5.3.5.2 Comparison of Simulated Developments 295
5.3.5.3 Robust requirement Formulation 302
5.3.6 Conclusions on Coordinated Optimization Experiments 303
5.3.6.1 Philosophical Overview 303
5.3.6.2 Discussion on Multi-Level Optimization Methodology 304
5.3.6.3 Overview of Coordinated Optimization Benefits 305

Chapter 6 Proof of Concept 307

6.1 Introduction 307
6.1.1 Purpose of the Case Study 308
6.1.1.1 Criteria for Experiment 4 309
6.1.1.2 Proof of concept for the overall methodology 310
6.1.2 Introduction of the Case Study 310
6.1.2.1 Novelty of the Architecture and its Failure Scenarios 312
6.1.2.2 Intricate Sizing Process 312
6.1.2.3 Wide Architectural Design Space 313
6.1.2.4 Large numbers of Subsystem Trade-offs. 313
6.1.2.5 Broad Scope of the Power Architecture 314
6.1.2.6 Description of the Design Problem Considered in the Test Case 314
6.1.3 Limitations and General Observations on the Case Study 315
6.1.4 Design Problem Formulation for the Proof of Concept 315
6.2 Preparation of the Design activities 318
6.2.1 Introduction of the Modeling Framework 319
6.2.1.1 Environment for the SySML Definition of Architecture Concepts 319
6.2.1.2 Environment for the Analysis Model Construction and Execution 321
6.2.2 Mathematical Analogy for Conceptual and Analysis Models 322

6.2.3 Mission Requirements Definition 324
6.2.4 Preparation of Subsystem Knowledge 326
6.2.4.1 Identification of Subsystem Types 326
6.2.4.2 Negotiation of Modeling Interfaces 336
6.2.4.3 Definition of Subsystem Sizing Models 344
6.2.4.4 Conclusions on Subsystem Knowledge Preparation 348
6.3 Definition of the Baseline Architecture 350
6.3.1 Overview of the Architecture 351
6.3.1.1 Description of the Architecture in Normal Operation 351
6.3.1.2 Failure configurations 355
6.3.2 Architecture Concept Definition in SysML 360
6.3.2.1 Definition of the Architecture Concept 362
6.3.2.2 Observations on Workload Implied by the Architecture Definition 372
6.3.2.3 Conclusions on Architecture Concept Definition 373
6.4 Description of the Builder 375
6.4.1 Conceptual Role of the Builder 376
6.4.2 Implementation of the Routine 377
6.4.3 Observations on the Builder Implementation 379
6.5 Architecture Analysis Overview 380
6.5.1 Overview of the Analysis Model and Introduction of Default Components 381
6.5.1.1 Implementation of the Architecture-Level Optimizer 382
6.5.1.2 Mission Block 383
6.5.1.3 ShowComponents Block 384
6.5.1.4 SystemSynthesis Block 386
6.5.2 Representation of the Mission 387
6.5.3 Implementation of Functional Relationships 389

6.5.4 Observations on the Model Composition 391
6.5.5 Assessment of the Practicality of the Analysis Model..... 394
6.5.6 Observation on the Architecture Analysis Model Setup Time 395
6.6 Architecture Analysis and organization of a Trade-Off 396
6.6.1 Modeling Refinement Loop 397
6.6.1.1 Adding a Missing Interaction..... 397
6.6.1.2 Modification or Specialization of Sizing Models 400
6.6.2 Feasibility Assessment Loop 401
6.6.2.1 Requirement Sensitivity Analysis (or Detecting the Constraining Requirements) 402
6.6.2.2 Negotiation of Functional Requirements in Conceptual Design 403
6.6.3 Optimization loop 404
6.6.3.1 Subsystem Sizing Optimization 404
6.6.3.2 Architecture Concept Optimization 407
6.6.4 Preparation of Architecture Developments..... 415
6.6.4.1 Formulation of the Subsystem Design Problem – an Example 416
6.6.4.2 Discussion of the Automated Subsystem Problem Formulation 421
6.7 Conclusions on the Model-Based Architecting Methodology..... 423
Chapter 7 Conclusions and Future Work 425
7.1 Overview of the Scientific Reasoning 425
7.2 Overall Conclusions on the Methodology 429
7.2.1 Complete Architecting Process..... 429
7.2.2 Provision of Deep and Flexible Architecture Analyses 430
7.2.3 Exploration of Subsystem Development Strategies..... 430
7.2.4 A Collaborative Framework 431

7.3 Future Work 432
7.3.1 Development of Monetary Objective Functions for Subsystem Developments 432
7.3.2 Methodological Augmentation of Model-Based Architecting Methods 432
7.3.3 Capturing Transient Behavior 433
7.3.4 Visualization of Analysis Results 433
7.3.5 Industrial Implementation 435
7.4 Final Words 435
Appendix A Quantification of boundary function specifications 436
Appendix B Electric Fan performance and sizing model 499
Appendix C Sizing the Power Plant 618
Appendix D Sizing the Electric Generators 623
Appendix E Sizing the Electric Buses 626
Appendix F Battery performance and sizing model 632
Appendix G Sizing scenario compacters 649
Appendix H Scriptwrapper Generator 653
Appendix I Architecture Model Builder – Code 658
Appendix J Framework User Guides 695
Appendix K System Synthesis Block 717
References 718

Index of Terms 725

Vita 727

List of Tables

Table 1: Physics-based MA 108
Table 2: Functional-based MA 108
Table 3: MA implementation of concept 1 109
Table 4: MA implementation of concept 2 109
Table 5: MA implementation of concept 3 109
Table 6: MA implementation of concept 4 109
Table 7: Structure concept 1 110
Table 8: Structure concept 2 111
Table 9: Structure concept 3 111
Table 10: Structure concept 4 111
Table 11: Comparison overview of Matrix/SysML based methods 115
Table 12: Methods, techniques and tools alternatives 134
Table 13: Criticality level description 160
Table 14: Specification of operating scenarios (phase 2 – step 1) 220
Table 15: Importing scenario transitions (phase 2 - step 2) 221
Table 16: Detection of possible mission sequence (phase 2 – step 3) 221
Table 17: Definition of mission states (phase 2 – step 4) 221
Table 18: Review of Hypotheses 228
Table 19: Description of boundary functions 235
Table 20: Flight conditions and durations 240
Table 21: Criticality level definition 241
Table 22: Overview of valid combinations 246
Table 23: Notional optimization priorities 259
Table 24: Front parameters for experiment 3.2 and 3.3 279

Table 25: Standard deviation applied in experiment 3.3 294
Table 26: Probability of improvements with Objective-Based design formulation 298
Table 27: Subsystem type general description..... 333
Table 28: Input / Output variables associated with the electric fan model..... 346
Table 29: Input / Output variables associated with the gas turbine model 346
Table 30: Input / Output variables associated with the VFG model..... 346
Table 31: Input / Output variables associated with the DCG model 347
Table 32: Input / Output variables associated with the bus models..... 347
Table 33: Dispatch Failure configurations..... 355
Table 34: Effects of dispatch failure configuration on non-propulsive generators 356
Table 35: Effects of dispatch failure configurations on propulsive generators 356
Table 36: Major Failure configurations 356
Table 37: Time cost of the architecture concept definition 372
Table 38: Example of the show array 385
Table 39: Input variables to the electric fan sizing model..... 416
Table 40: Functional Constraints for the electric fan..... 417
Table 41: Critical sizing requirements..... 418
Table 42: Trade-off equivalences for the electric fan..... 420
Table 43: JAR/FAR 25 climb requirements 438
Table 44: Manoeuvers implied by JAR/FAR 439
Table 45: Excess power requirements 441
Table 46: Degraded requirements from the FAR/JAR 442
Table 47: Requirements degradation profiles for study..... 443
Table 48: Dirty configuration factors 447
Table 49: Vehicle description..... 454
Table 50: Mission and requirements..... 455
Table 51: Thrust calculations for ground operation..... 458

Table 52: Take off calculation inputs 460
Table 53: Thrust requirement calculation for take off run 462
Table 54: TOCL1 thrust requirements calculations 463
Table 55: Phase 5 flight conditions 464
Table 56: Phase 5 maneuver thrust requirements calculations 464
Table 57: Phase 6 flight conditions 466
Table 58: Phase 6 maneuver thrust requirements calculations 467
Table 59: Phase 7 flight conditions 468
Table 60: Phase 7 manoeuvre thrust requirements calculations 468
Table 61: Phase 8 flight conditions 470
Table 62: Phase 8 maneuver thrust requirements calculations 470
Table 63: Phase 9 flight conditions 471
Table 64: Phase 9 maneuver thrust requirements calculations 472
Table 65: Phase 10 flight conditions 473
Table 66: Phase 10 maneuver thrust requirements calculations 473
Table 67: Phase 11 flight conditions 475
Table 68: Phase 11 maneuver thrust requirements calculations 475
Table 69: Phase 12 flight conditions 477
Table 70: Phase 12 maneuver thrust requirements calculations 477
Table 71: Req. for max thrust 480
Table 72: Req. for nominal thrust 481
Table 73: Functional requirements for thrust force 481
Table 74: Functional profile in normal operation 484
Table 75: Degradation profile I for the customer and technical loads 484
Table 76: Degradation profile II for the customer and technical loads 485
Table 77: Degradation profile III for the customer and technical loads 485
Table 78: Normal load levels for control 489

Table 79: Flows assigned to the control sub-functions 491
Table 80 : Normal control requirement profile 492
Table 81: Level II control requirement profile 492
Table 82: Level III control requirement profile 493
Table 83: Load required by the primary control function 494
Table 84: Loads required by the secondary control function 494
Table 85: Environment conditioning normal operation 495
Table 86: Environment conditioning level I operation 496
Table 87: Environment conditioning level II operation 496
Table 88: Environment conditioning level II operation 496
Table 89: Overview of input/outputs 502
Table 90: Typical flight conditions 564
Table 91: Thrust limits for sizing model 564
Table 92: Notional optimization strategies 565
Table 93: Efficiency used in bus sizing 627
Table 94: Bus efficiencies 627
Table 95: Example with redundant scenario 649
Table 96: Example reduced 650
Table 97: example of "address" matrix 650
Table 98: Example of induced requirements 651
Table 99: Example of de-compacted functional requirements 651

List of Figures

Figure 1: Design phases 15
Figure 2: Changes in developments [12] 18
Figure 3: Feeding information forward [11] 20
Figure 4: Serial vs. concurrent engineering 21
Figure 5: Disciplinary perspectives by C.W. Miller [13] 23
Figure 6: Clustered-concurrent development framework 26
Figure 7: Clustered-concurrent engineering approach 27
Figure 8: Curves and structures of the Osaka Airport 34
Figure 9: MDA based approach 36
Figure 10: IPT based approach 37
Figure 11: Time associated with performance of integrated analysis 40
Figure 12: Illustration of assumptions in the context of modeling (part 1) 44
Figure 13: Illustration of assumptions in the context of modeling (part 2) 44
Figure 14: Representation of analysis under uncertainty 45
Figure 15: Utility implied by a target 60
Figure 16: Utility associated to an objective 61
Figure 17: Notional representation of conceptual design 63
Figure 18: Notional subsystem design after conservative conceptual design predictions 64
Figure 19: Notional subsystem design after optimistic conceptual design predictions ... 65
Figure 20: Comparison of IPT and MDA 67
Figure 21: IPT MDA based approach 68
Figure 22: Activities constituting ASA conceptual design 70
Figure 23: Requirement definition phase 71
Figure 24: Function Family Tree principle 73

Figure 25: Notional functional breakdown of aircraft functions[28] 74
Figure 26: FAST diagram principles 75
Figure 27: Example FAST diagram 76
Figure 28: Objective tree 80
Figure 29: Notional objective tree for an aircraft development program 80
Figure 30: Architecture concept definition in conceptual design 83
Figure 31: Physical breakdown 86
Figure 32: Notional geometric decomposition 87
Figure 33: Notional functional breakdown[28] 87
Figure 34: ATA chapters 89
Figure 35: Role of architectural perspectives in the analysis activities 91
Figure 36: Notional Matrix of Alternatives 92
Figure 37: IRMA exemple [Engler et al 2007] 93
Figure 38: Compatibility matrix 94
Figure 39: SysML diagram 95
Figure 40: Block definition diagram example 96
Figure 41: Interaction graph 98
Figure 42: Self-interaction matrix 98
Figure 43: Directed graph 99
Figure 44: Directed interaction matrix 99
Figure 45: Networks with explicit connections 101
Figure 46: Models' I/O ports 101
Figure 47: Genetic Network String of the example architecture 102
Figure 48: Internal block diagram example 103
Figure 49: Informal symbols for subsystems and relationships 106
Figure 50: Informal representation of concept 0 106
Figure 51: Notional concepts for test case 106

Figure 52: SysML description of architecture structure 112
Figure 53: Role of modeling and analysis in conceptual design 116
Figure 54: Elements of the QFD[28] 117
Figure 55: Example DSM 121
Figure 56: DSM in matrix form 121
Figure 57: Fixed point iteration for MDA convergence 123
Figure 58: Optimizer based decomposition 124
Figure 59: Sample MDO 125
Figure 60: All in one optimization setup 125
Figure 61: Collaborative optimization setup 126
Figure 62: ATC hierarchy 127
Figure 63: derivative analysis process 130
Figure 64: Types of change propagation 131
Figure 65: Technology Mapping Methodology Roadmap 135
Figure 66: MA choices for Technology Mapping Methodology 136
Figure 67: Matrix of alternative for Fuel Systems [68] 137
Figure 68: Interaction matrix for fuel systems [67] 137
Figure 69: MA choices for Quantified relationship matrices 138
Figure 70: STeP process 139
Figure 71:MA choices for STeP 140
Figure 72: Integrated sizing and performance process 141
Figure 73: MA choices for a simulation framework for aircraft power management 142
Figure 74: Architecting framework composition 145
Figure 75: Architecting Team role 145
Figure 76: Architect role 146
Figure 77: Expert role 146
Figure 78: MDA role 146

Figure 79: Organization of the overall process 149
Figure 80: Process overview - Mission requirement definition 152
Figure 81: Implementation of functional breakdown 155
Figure 82: Criticality example 159
Figure 83: Definition of the scenario stereotype 162
Figure 84: Process overview - Subsystem knowledge preparation 164
Figure 85: Identification of function flows 169
Figure 86: Functional definition of functional types 170
Figure 87: Parallel between functional relationships and sizing relationships 171
Figure 88: Notional functional relationship 172
Figure 89: Classification of variable flows 173
Figure 90: Definition of the generic block for function flow 173
Figure 91: Characterization of variable flow with an IBD (example 1) 174
Figure 92: Example of function flow types 176
Figure 93: Example of subsystem types 176
Figure 94: Example with generator and motor model interfaces 176
Figure 95: Different requirements in different scenarios 177
Figure 96: Example of variable connections in array 178
Figure 97: Subsystem front with system-level objective 191
Figure 98: Observation of different optimization strategies 192
Figure 99: Process overview – Architecture concept definition 198
Figure 100: Overview the architecture concept definition process 201
Figure 101: Step 1 in defining the architecture composition 203
Figure 102: Step 2 in defining the architecture composition 203
Figure 103: Introducing configurations 205
Figure 104: Definition of configuration “Normal” 206
Figure 105: Definition of configuration “Engine2out” 207

Figure 106: Definition of mission sequences 208
Figure 107: Allocation Matrix for scheduling configurations 211
Figure 108: Process overview – Automated model construction 212
Figure 109: SMDA composition 213
Figure 110: SMDA structure 213
Figure 111: Overview of the model construction process 215
Figure 112: Importing the architecture composition (phase 1 - step 1) 217
Figure 113 : Setting up the optimization connections (phase 1 –step 2) 218
Figure 114: Returning subsystem attributes to system synthesis (phase 1 – step 3) 218
Figure 115: Simple mission sequence 220
Figure 116: Structure builder process (phase 3) 224
Figure 117: Identification of functional relationships (phase 3 – step 1) 225
Figure 118: Retrieving the flows of variables (Phase 3 – step 2) 226
Figure 119: Definition of the connection (phase 3 - step3) 226
Figure 120: Overview of fundamental research questions and objectives 229
Figure 121: Overview of the philosophical process 231
Figure 122: Experiment 1 overview 232
Figure 123: Functional breakdown of a commercial transport aircraft 234
Figure 124: Mission sequence overview 236
Figure 125: Phase 1 236
Figure 126: Phase 2 236
Figure 127: Phase 3 236
Figure 128: Phase 4 237
Figure 129: Phase 5 237
Figure 130: Phase 6 237
Figure 131: Phase 7 237
Figure 132: Phase 8 237

Figure 133: Phase 9 238
Figure 134: Phase 10 238
Figure 135: Phase 11 238
Figure 136: Phase 12 238
Figure 137: Distribution of flight phases 240
Figure 138: Phase 1, 2 and 3 failures mission sequences 242
Figure 139: Phase 4 failures mission sequences 242
Figure 140: Phase 5 failures mission sequences 243
Figure 141: Phase 6, 7 and 8 failures mission sequences 244
Figure 142: Phase 9 and 10 failures mission sequences 245
Figure 143: Experiment 2 overview 250
Figure 144: Graphical representation of the electric ducted fan 251
Figure 145: Thrust capability as a function of fan and motor sizes 256
Figure 146: Attributes scaled by requirements 259
Figure 147: Overview of internal trade-offs 261
Figure 148: Experiment 3 overview 264
Figure 149: Simple architecture for example 3 271
Figure 150: Notional Pareto front 273
Figure 151: Notional non-convex Pareto front 273
Figure 152: Convex problem 279
Figure 153: Hybrid problem 279
Figure 154: Double non-convex problem 279
Figure 155: Overview of topologies 281
Figure 156: Design details associated with topology 282
Figure 157: Illustration of the Pareto threshold 283
Figure 158: Weight breakdown 283
Figure 159: Comparison of convex problem topologies 285

Figure 160: Pareto discontinuity for AiO approach 285
Figure 161: Pareto discontinuity for CoOp approach 286
Figure 162: Blind spots of the CoOp in the non-convex problem 288
Figure 163: Architecture optimal solution in concave section 289
Figure 164: Target-based design problem 292
Figure 165: Objective-based design problem 293
Figure 166: Development simulation process 295
Figure 167: Example simulated developments 296
Figure 168: Probability of success in achieving an architecture objective 300
Figure 169: Spread of improvements resulting from objective-based design problems 301
Figure 170: Overview experiment 4 309
Figure 171: N2A Turbo-Electric modification presented by Felder [80] 311
Figure 172: Turbo-Electric aircraft concept for regional jets by Mark Waters [81] 311
Figure 173: The “next DC-3” concept by ASDL[82] 311
Figure 174: Overall process implied by the methodology 318
Figure 175: Role of MagicDraw 320
Figure 176: Conceptualization of model components 322
Figure 177: Process Overview – Mission requirement definition 324
Figure 178: Implementation of mission requirement definition 325
Figure 179: Process Overview - Subsystem knowledge preparation 326
Figure 180: Example declaration of subsystem type 327
Figure 181: Example declaration of functional ports 328
Figure 182: Example declaration of attributes 330
Figure 183: Example declaration of the priority factors 331
Figure 184: Example designation of the sizing model in the AS library 332
Figure 185: SysML declaration of subsystem types for test-case 335
Figure 186: Process illustration – Negotiation of modeling interfaces 336

Figure 187: General description of functional flow structure 337
Figure 188: Definition of the generic function flow 338
Figure 189: Pre-definition of functional flows 339
Figure 190: Example of functional flow diagram 339
Figure 191: Example naming convention 341
Figure 192: Definition of E_FFAC_power flow 342
Figure 193: Definition of E_VFAC_power flow 342
Figure 194: Definition of PropulsivePower flow 343
Figure 195: Definition of mech_power flow 343
Figure 196: Process illustration – Definition of subsystem models 344
Figure 197: Process Overview - Subsystem knowledge preparation 350
Figure 198: Overview of architecture 1 without failures 353
Figure 199: Arch.1 - Park configuration 354
Figure 200: Arch.1 - Taxi Configuration 354
Figure 201: Arch.1 - Take-off configuration 354
Figure 202: Arch.1 - Airborne configuration 354
Figure 203: Overview of configurations associated with failure 1 358
Figure 204: Overview of configurations associated with failure 2 359
Figure 205: Process Overview – Architecture Concept Definition 360
Figure 206: Overview of the Architecture Concept Definition activity implementation 360
Figure 207: Definition of Architecture 1 composition 362
Figure 208: Listing of architecture configurations 363
Figure 209: Park configuration - Architecture 1 364
Figure 210: Normal operation take-off (and landing) configuration - Architecture 1 365
Figure 211: Failure 1a take-off (and landing) configuration - Architecture 1 366
Figure 212: Failure 2a take-off (and landing) configuration - Architecture 1 367
Figure 213: Mission sequences - Architecture 1 369

Figure 214: Allocation Matrix for dispatch failures 370
Figure 215: Allocation Matrix for major failures 371
Figure 216: Allocation Matrix for normal operations..... 371
Figure 217: PERT chart of the architecture concept definition 372
Figure 218: Process Overview – Automated construction of the analysis model 375
Figure 219: Overview of the architecture analysis builder implementation..... 376
Figure 220: Overview of builder classes 377
Figure 221: Builder action group in MagicDraw..... 378
Figure 222: Analysis model composition overview 381
Figure 223: Representation of all possible mission sequences 388
Figure 224: Implementation of a functional relationship 389
Figure 225: Close up view on model 392
Figure 226: Process overview - Architecture Analysis..... 396
Figure 227: Redefinition of functional flows..... 399
Figure 228: Integration of a new functional relationship..... 400
Figure 229: Optimizer setup for coordinated optimization..... 406
Figure 230: Architecture-level optimizer topology for architecture 1 407
Figure 231: Architecture concept optimization loop 409
Figure 232: Architecture 2 – Architecture composition 410
Figure 233: Architecture 2 – Take-off configuration (non-prop. subsystems)..... 411
Figure 234: Architecture 2 – Airborne configuration (non-prop. subsystems) 411
Figure 235: Architecture 2 - Take-off configuration (propulsive subsystems) 412
Figure 236: Architecture 2 - Airborne configuration (propulsive subsystems)..... 412
Figure 237: Overview of architecture 2 analysis model 413
Figure 238: Overview of the architecture solution implementation 415
Figure 239: Overview of the scientific reasoning..... 428
Figure 240: Overview of the proposed architecting process 429

Figure 241: Example Displaying Subsystem Attributes 434
Figure 242: Example Displaying Functional Flow Attributes 434
Figure 243: Degraded requirements from the FAR/JAR 444
Figure 244: Polar curves for the dirty configurations 447
Figure 245: Take-off sequence 450
Figure 246: Thrust requirements calculator layout 456
Figure 247: Overview of take off requirements 459
Figure 248: Convergence process toward thrust requirements 462
Figure 249: Customer functionalities breakdown 482
Figure 250: Control function breakdown 488
Figure 251: Hydraulic load profile for an A320[89] 490
Figure 252: Notional representation of the ducted electric fan[77] 499
Figure 253: Logic of the electric ducted fan sizing model 500
Figure 254: Structure matrix of the electric ducted fan sizing model 501
Figure 255: Stations used in the analysis 504
Figure 256: Incoming flow velocity triangle 513
Figure 257 : Mass flow through fan area 514
Figure 258: Baseline fan geometry 515
Figure 259: Notional fan map 516
Figure 260: Matching problem 525
Figure 261: Operational fan/motor matching 527
Figure 262: Constrained fan map 531
Figure 263: Duct and motor constraints solving 533
Figure 264: Overview Thrust Capability Model 535
Figure 265: Typical efficiency profile 537
Figure 266: Typical engine characteristics at max thrust 538
Figure 267: Constrained fan map at low altitude 539

Figure 268: Operational regions 539
Figure 269: Topology of the thrust capability 540
Figure 270: Constraint in the ro-Afan space 543
Figure 271: Notional fan area constraint 544
Figure 272: Identification of the critical sizing scenario 544
Figure 273: Engine pre-sizing process 546
Figure 274: Penalty function discontinuity 548
Figure 275: Fragmentation of design space 549
Figure 276: Issues with single fitting 551
Figure 277: Thrust capability evolution by zones 554
Figure 278: Boundary fitting error 556
Figure 279: Typical behavior of efficiencies at idle and max thrust 558
Figure 280: Typical behavior of engine mass 559
Figure 281: Surrogate model building process 560
Figure 282: Predicted responses with error representation 561
Figure 283: Sized designed scaled by requirements 565
Figure 284: Overview of internal trade-offs 566
Figure 285: Illustration of the inverse flow function regression 568
Figure 286: Description of the power plant subsystem 618
Figure 287: Battery Model Overview 632
Figure 288: Representation of the battery 633
Figure 289: Typical voltage drop for a lead battery 634
Figure 290: Recharge rates 636
Figure 291: Flowchart for the battery performance model 637
Figure 292: Battery optimizer flowchart 639
Figure 293: Example load profile 640
Figure 294: Performance of an optimal battery 641

Figure 295: Performance of an undersized battery 642
Figure 296: Performance of an oversized battery 643

Summary

After many decades of abundant government funding following the cold war, the commercial aerospace industry is transitioning to a new phase of autonomy where human and capital resources are increasingly limited. In this context, the investments related to subsystem technological developments are increasingly outsourced to suppliers. These first tier contractors are often shared among airframers. As a result, the access to new technologies is no longer the main factor for product differentiation. On the other hand, optimizing the aircraft system architecture for superior integrated performance of available technologies is more than ever a core competency to ensure product competitiveness. This highlights the need to consider architecting activities in conceptual design. Traditional system engineering methods, expected to support this capability, were more appropriate to coordinate progressive developments of subsystems within a fixed framework, rather than the exploration of architecture concepts and strategies.

In order to address this gap, a model-based architecting methodology is proposed. This methodology addresses limitations of traditional system engineering methods. The system engineering methods allow for the organization of developments, but are limited in their ability to explore and analyze new architectures. Therefore, the proposed methodology allows for expedient analysis of architectural concepts and the establishment of subsystem development strategies. Its process is implemented by an architecting team composed of subsystem experts and architects to respect existing industrial structures. The methodology uses the SysML language as a visual and organized means to define architectures. Using meta-modeling techniques, this definition is translated into an analysis model which automatically integrates subsystem analyses in a fashion that represents the specific architecture concepts described by the team. The resulting analysis automatically sizes the subsystems composing it, solves and optimizes the subsystem trade-offs, and synthesizes their information to derive architecture-level

performance. The subsystem optimization is facilitated using the Coordinated Optimization method proposed in this dissertation. This method proposes a multi-level optimization setup. An architecture-level optimizer orchestrates the sizing of subsystems in order to optimize the aircraft as whole while maintaining subsystem expertise boundaries.

The proposed overall methodology enables the exploration of the architectural design space, via the acceleration of the analysis cycle of architecture concepts. The acceleration is primarily performed via the implementation of architecture analysis based on the automated integration of subsystem models. Architecture models which used to necessitate weeks of preparation can now be setup in hours. With this spectacular acceleration of the analysis setup, more architecture concepts can be explored, and better architectural solutions identified.

The benefits of the proposed methodology are not limited only to the exploration of the architecture concepts. It also formulates design requirements for subsystem developments. This formulation provides objective functions based on subsystem domain parameters which will guide their optimization toward architecture-level goals. This approach has been demonstrated to provide a formidable advantage in terms of the integrated performance compared to traditional system engineering approaches. This advantage is perceptible both by higher performance of the architecture after development and by a significant improvement in the probability of success in meeting system performance levels.

As a result this methodology empowers the system architects in their ability to identify the best architecture concepts and to perform subsystem integration in the best and most effective way possible.

Chapter 1

Motivation

1.1 General Introduction

Over the past 20 years, bankruptcies and merging of major commercial aircraft manufacturers have decreased the number of competitors in this field. The result is a concentration of the market around two major competitors: Airbus and Boeing. In parallel with this market concentration, the air transportation business has continued to grow. This growth has been accompanied by the evolution of the demand in terms of aircraft performance and functionality requirements. To satisfy and capture this growth, Airbus and Boeing continuously adapt their product lines by accelerating the frequency and duration of new aircraft developments. The risk and the resources required to keep up with this tendency led them to increased risk sharing policies with their subcontractors. Consequently, some technological developments of new aircraft programs are not conducted "in house", but rather are externalized to former suppliers. These technological partners use their specific system expertise to propose new technologies optimizing the performance of their individual systems rather than at the aircraft level.

In this context, integration and architecting activities are undergoing a revolution. The variety of new solutions resulting from the involvement of technological partners forces a paradigm change in system architecting activities. Rather than leading system modification directed by aircraft optimization, the architect leads the integration of individually optimized systems within a modified baseline architecture.

System level optimization tends to increase the number of technological assets available to the architect. However, the integration of these new assets does not necessarily guarantee optimized aircraft characteristics. As a result, the effects of system level technology on aircraft characteristics become less clear. At the same time, the

architects' understanding of new technologies is complicated by multi-organizational developments. In this situation, the traditional approach of an individual architect becomes increasingly inappropriate to integrate new developing technologies.

1.2 Aircraft System and Architectures

Unlike most industrial products, a commercial aircraft requires a degree of technological sophistication and complexity which differentiates it from most. Because of this, the public perception of aerospace engineers has classified us as people with extraordinary technical capabilities (or at least higher than average). If you read this thesis, you probably know that we are, unfortunately, not cleverer than our counter-parts in others industries. As a result, the extraordinary technical challenges offered by an aircraft development are addressed using a sophisticated organizational structure. This organization allows the subdivision of this complex problem into simple, or at least technically manageable, projects. The more complex the technical problem is, the more subdivided it becomes. This subdivision practice is defined, ruled, and managed by systems engineering. As a result, an aircraft development is really a network of developments including engine development, fuselage development, control systems development, cabin development, etc…This breakdown structure allows the engineers to deal with "simpler" problems toward the realization of a technologically complex ensemble. In other words, systems engineering facilitates the design of complex systems by translating them into interdependent manageable designs of "simpler" systems.

From this observation we can see two levels emerging: the complex system level (the aircraft) and the "simpler" system level (pumps, generators, electrical wires, etc…). In this thesis the complex system level will be referred to as the **architecture** and the "simpler" system as **subsystem** or **component**.

Although system engineering facilitates the management and cohesion of the component level developments, the technical complexity at the architecture level remains.

As a result, the definition of an aircraft system architecture remains a great challenge especially in the initial phases of design.

Rather than facing this great challenge frontally, the industry has opted for a slower and more frugal approach from a development point of view. Instead of reconsidering the architecture in the light of the latest technological opportunities, classical architectures have been modified primarily through the substitution of an existing component by its "next generation" (read new technology) equivalent.

This derivative approach gave birth to what has been referred to as the "family of systems" design. This term, which originated in the military organizational field, is employed to describe a situation where decision makers have to redefine the organization of an ensemble of assets inter-related operationally and where the structure of relationships is fixed. This term is a somewhat humoristic metaphor, where the family members refer to the components and the family as a whole to the architecture. This metaphor illustrates well the current architecture design paradigm from two perspectives. The first is the fact that one is born in a family and *de facto* does not have the choice to decide on its constitution ("You choose your friends but not your family). The second perspective comes from the fact that in a stereotypical family, every one has his role and position. In situations, one of the family members gets substituted by its equivalent, but the structure of the family does not change.

1.3 Evolutions of Aircraft Functionalities

Until recently the aircraft system architecture has been approached as a family of systems. This family provided a classical breakdown of the aircraft into components associated with specific functions. This approach enabled compartmented studies focusing on individual component early in the design process. Doing so, defines decoupling and simplifying interfaces between subsystems as pre-defined boundaries and breaks down the operating environments for each component into what is referred to as a

functional specification [1]. This approach is employed to simplify technical studies, and to minimize the exchange of information outside the perimeter of the component. This simplification limits the exchange of information between experts and organizations.

Hence, considering the architecture as a framework, where components are associated with a specific mission (or functional specification), has offered many years of opportunity in system based optimization. In this context, improvements at the component level typically "granted" improvement at the aircraft level. However, the industry is beginning to realize that this approach is now returning limited improvements. Consideration of the architecture as a new dimension in the design space has now become a necessity.

This evolution can be explained by two main factors:

- The functionalities of an aircraft are increasing both in number and complexity
- The optimization of aircraft subsystems has fostered the development of technologies requiring a modification of the architecture

The expectation on aircraft functionality has evolved dramatically. The "hosting" functionality of commercial aircraft has considerably improved. The time where flying was an adventurous experience is now long gone. The adventure has now been substituted by comfort, and safety. The In Flight Entertainment systems (IFE) have become a central requirement for airline companies, with a power consumption that rose significantly over the past few generations. In parallel, the cabin air quality (mass flow, pressure, and humidity) is also evolving.

Much like the "hosting" functionality, functions like safety have also evolved drastically. New functional requirements like fuel tank inerting (removing oxygen from fuel tank to limit risk of explosion) have been introduced, imposing new subsystems and new energy loads on the architecture.

1.4 Evolution of Aircraft System Technologies

In parallel with those functional evolutions, the component optimization process has progressively changed the role of electric power within the architecture. Goodbye archaic mechanical/hydraulic controllers, welcome fly-by-wire, Full Authority Digital Engine Control, etc… The criticality of these new applications of electric power has initiated some evolutions in the power architecture, by imposing new segregation rules between loads. But the most predominant evolution has been introduced by the "more electric technologies" which substitute subsystems traditionally operating on pneumatic or hydraulic energy with their electric equivalent. These new components relying on a different source of power than their counterparts, force a drastic change in the power architecture.

The traditional commercial aircraft power architectures were designed at a time when non-propulsive power was considered as "secondary" with an almost negligible impact on overall vehicle performance. Consequently, the design trades in conceptual design hardly ever focused on this aspect. With the current evolution in aircraft functionality and system technology, ignoring these trades in early developments has led to several technical difficulties.

Postponing architectural trades does not necessarily leave enough design space to accommodate new requirements. Physical space available for installation, or power availability for operation of the new, larger solutions necessary to satisfy new requirements are no longer sufficient. Consequently, what used to be negligible in the past no longer is. The increased pneumatic off-take at the engine level induces significant penalties in terms of installation, the increase in electric power affects engine mechanical loading and modifies its operability and fuel burn. Therefore, using a fixed architecture as an answer to changing aircraft functionality and new technologies is no longer an appropriate solution.

The aggregation of all these effects, evolutions of functional requirements, introduction of new aircraft functions, induction of new segregation rules and transformation of systems with new technologies has forced the industry to reconsider its architectural approaches.

1.5 Development of Complex Systems within a Multi-Organizational Environment

In order to decrease the risks related to the development of a new aircraft, the technical and financial burden of the development is shared over multiple organizations (**outsourcing of developments** [2]). Historically, the research and development were performed primarily by the airframer, while suppliers were limited mostly to the detailed definition of the small subsystems along with its production. Several factors have forced a change in this paradigm.

Despite the apparent equilibrium of the Airbus/Boeing duopoly (probably due to its relative longevity), both actors believe to be under continuous threat on their market. Commercial aircraft design and manufacturing is a business where product differentiation is mostly focused on performance (operating cost). In this market technology is can provide a large competitive advantage which may suddenly modify the equilibrium in this monopoly. In order to maximize their expansion, Airbus and the Boeing Commercial Aircraft division (BCA) are caught into a "leapfrogging" strategy[3]. A "**leapfrogging**" is a marketing term used to characterize a strategy where the follower attempts to surpass the leader via innovation [4-5]. This market strategy has forced Airbus and BCA to accelerate their development cycles by:

- Starting new aircraft development programs more often.
- Reducing the duration of the program.

The fact that this approach was adopted simultaneously by both Airbus and BCA makes this strategy both necessary to their survival and threatening to their financial

balance. Therefore, it is now necessary for both of them to keep up with this technological escalade in order to maintain their market share.

In order to alleviate financial risks associated with this situation, both Airbus and BCA have started to redefine their development paradigm. In order to decrease the investment load necessary to keep up with new development cycles, technical responsibility along with financial participation in the program was shared with former suppliers. This evolution transformed the relationships between the airframer and its former suppliers. The suppliers are not longer sub-contractors bidding to perform a preconceived task or receiving initial funding from the airframer to develop specific solutions. The relationship is now evolving toward a technological partnership where the airframer and the former suppliers are now investing together in the R&D of the technological portfolio necessary to the development of the aircraft. As a result, entities external from the airframer are involved earlier in the development process and are responsible for the development of larger system perimeters[6] [7].

This evolution must also be considered in a context where the system architecture is changing. The technological partnerships have been perceived as an opportunity to share the challenges related to the definition of the architecture. This has been done by isolating architectural changes within a work package (large subsystem perimeter). To illustrate this strategy, one could consider the perimeter of responsibility of Hamilton Sundstrand in the Boeing 787 program. The 787 is the first large commercial aircraft using electrically powered air conditioning. This new technology, relying on an electrically powered compressor rather than pneumatic energy extracted from the turbofan, requires profound architectural modifications. In order, to compartmentalize the complexity induced by this new technology, the entire electrical architecture along with the air conditioning subsystem was placed under the responsibility of Hamilton Sundstrand. In a similar fashion, the responsibility of the structure for the wing and fuselage using composite material was delegated to Mitsubishi Heavy Industries for the

wing and Vought for the fuselage. This approach has indeed facilitated the development of the new architecture employed by the 787 in the sense that the work load necessary to perform the detailed design activities could be delegated to multiple organizations. But recent delays in the development program of the 787 seem to indicate significant difficulties related to the integration of the systems developed by the partners.

When the development of the architecture is split into multiple system developments, one of the greatest technical difficulties results from the uncertainty on the interaction between those systems. The larger a design perimeter is, the more complex are its interactions with other perimeters. Consequently, if the definition of a large perimeter is under the responsibility of a partner, his developments will strongly impact the definition of the rest of the architecture. In the same fashion, he will be easily impacted by changes outside its perimeter.

Some of those risks can be alleviated by good collaboration among technological partners. Traditionally, this collaboration exists between the aircraft and the engine manufacturer. For many years, airframers and engine makers have maintained a close collaboration throughout the aircraft development program. Discussion between the two parties often starts very early in the design process (from early feasibility phases) and the exchange of information characterizing the behavior of the engine is continuous. The engine's central role within the power architecture and the fact that it is a primary driver of aircraft performance, forced the establishment of this relationship. But this type of relationship hardly ever existed between the airframer and other suppliers due to the fact their system was not "central" (in other words, had a relatively low impact on the rest of the architecture) and did not closely relate to overall performance. Now that design perimeters have grown larger, functional priorities have evolved, and technologies have changed, the development practices must also adapt to the increased importance of formerly minor subcontractors, now major technological partners.

Beyond the organizational process required to monitor architecture configurations, or system level requirements, there is also a strong need to anticipate these changes. Instead of dealing with difficulties as they occur, it would be preferable to foresee technical difficulties, and identify optimization guidelines for each perimeter. In order to develop a subsystem relying on new technologies, time is required to perform the R&D necessary to improve the subsystems within the architecture. The earlier the functional specifications are defined, the more stable the specifications remain during the development, and the greater the opportunity for the subsystem designer to develop its technology optimizing its design to the aircraft benefit. In this context, successful collaboration requires a good understanding of both the architecture and the expected characteristics of the system that can be provided by the technology partner.

1.6 Conclusion

The simultaneous evolution of airline needs for a profitable aircraft to operate, and the increase of standards in air travel (both in terms of safety, comfort and flight entertainment) have forced the airplane makers to reconsider the way they design and develop their aircraft. The classical architecture, progressively matured and optimized over the past decades, is unable to incorporate the technologies necessary to sustain the evolution of needs.

As a result, radically new architectures are currently being considered by airframers in their new aircraft developments. But considering new aircraft system architectures is a considerably complex task, which requires a great deal of preparation in the conceptual phases. This preparation is necessary to substitute for the lack of general knowledge on the architecture required to optimize the aircraft concept, and guide system developments. In a context of dire competition between Airbus and the Boeing Company (without even mentioning potential future competitors in China, India, and/or Russia), mastering the ability to consider new AND optimized architectures, while maintaining a

solid industrial plan for technology partners in the aircraft development program, is a key strategic ability. This thesis will attempt to formulate a solution to this need.

Chapter 2

Problem Definition

In order to understand the challenges evoked in the previous chapter, one must come back to the meaning of "design". Therefore this chapter will be dedicated to the analysis of the process of designing a complex system. In this analysis, this chapter will describe the widely promoted idea of concurrent engineering. From this description we will observe how this idea and its founding principles influenced industrial practices in aircraft developments.

Given this industrial context, we shall focus the analysis on the conceptual design phase, which sees the genesis of the Aircraft System Architecture (ASA). This phase, rich in opportunities and technical challenges, will be analyzed. This analysis will first describe its structure in terms of activities and participants. Then, it will focus on how the conceptual design drives the system engineering-based industrial development in subsequent phases. The definition of the focused problem addressed by this thesis will be derived from these observations.

These topics will be discussed in the following sections:

- Complex System Design and its Development Cycle
- Design in an Industrial Context
- Challenges in Applying Concurrent Engineering Approaches to Architecture Conceptual Development
- Linking Conceptual Design to Clustered Concurrent Preliminary and Detailed Design
- Research Objectives and Research Questions

2.1 Complex System Design and its Development Cycle

The development of an aircraft and its architecture is a design activity. In this thesis, the term architecture refers to the ensemble of subsystems composing the aircraft. As a result, we can say that the development of the architecture requires the design of an ensemble where each element awaits its own design. These cascading levels of design require a clear understanding of the process implied by the term "design". Therefore, before going into the industrial aspect of the problem, it is important that some basic notions related to design are defined as they will eventually become the building blocks of the hypothesis proposed in this dissertation. Beyond the definition, we shall observe the activities structuring the development of a complex system.

2.1.1 **Defining "design"**

In my academic education, I have been presented to several definitions of **design**. Some of the most interesting ones are presented below:

The first definition is provided by Anderson [8].

> Design: The intellectual engineering process of creating on paper a flying machine that either meets certain constrain requirements and performance objectives […].

The second definition is a definition by Mc Govern [9].

> Design: Design is about creating something with a purpose

The third definition was provided by Paredis [10].

> Design: Design is the process of converting information about needs and requirements for a product/system into a complete specification of that product/system

Each of the quotes above brings a slightly different perspective on the term *design*. Anderson provides a definition directed specifically directed toward aircraft design. Mc

Govern provides a more generic definition, and Paredis adds some level of details on the nature of the outcome of the design process. But beyond the formulation differences, each definition describes design as a process with inputs and outputs. The input is presented as a form of motivation described as:

- "certain constrain requirements and performance objectives" Anderson
- "a purpose", Mc Govern
- "Needs and requirements". Paredis

In each of these formulations, we can see that the motivation is based on pre-formulated needs driving the design activities. Anderson breaks down the motivation into two categories: "performance objectives" and "constrained requirements". This breakdown provides a useful distinction between negotiable and non-negotiable needs. This distinction will play a central role in the development of the thesis.

The outcome of the design process is presented as:

- "a flying machine", Anderson
- "something", Mc Govern
- "a complete specification of that product", Paredis

The definitions commonly indicate that the outcome is a tangible solution to the need previously identified. The formulation by Paredis ("*a complete specification of that product"*), highlights the fact that the solution must provide a physical description of the systems.

The verbs describing the action implied by the design activity are:

- "creating" Anderson and Mc Govern
- "converting" Paredis

The term used by Paredis (*converting*) provides a simple description of the actions. Design is about converting needs into some form of a solution. The conversion process relies on the knowledge under the form of experience, organized information (models),

and intuitions. Based on these different forms of knowledge, the designer identifies a new, optimal and physically-feasible solution. The term "creating" used by Anderson and Mc Govern attempts to capture these dimensions. To summarize the analysis of the definitions presented above, the following definition is proposed:

> Design is the process of creating the "best" physical solution possible which can fulfil a set of requirements.

The **design problem** is constituted of requirements which qualify the non-negotiable needs from the stakeholders, and a value framework for evaluating the "goodness" of the proposed solution.

Based on this formulation, we recognize the format of an optimization problem. In this optimization problem the optimization variables X correspond to the definition of the "physical solution", and the objective function corresponds to the quantification of the physical solution value (V). The inequality constraints represent the fact that the capabilities (C) of the solution must match or exceed requirements (R) and the equality constraints characterize the attributes (Att), value and capability of the physical solution (designated by X) under the specified operating conditions (Spe). This optimization formulation of the design problem is presented in equation (1). When resolving a design process, we attempt to resolve this optimization problem by identifying X* (or optimal set) which correspond to the description of the "best physical solution which can fulfil a set of requirements".

$$\underset{X}{Max} \; V(Att) \qquad (1)$$

Subject to:

- $R \le C(X, Spe)$
- $Att = f(X, Spe)$

Outcome:

Description of physical solution: X^* and Att^*

Note: The asterisk will be used to characterize sets of variables which designate an optimal solution to the optimization problem. In this context, these parameters represent the outcome of the design process

2.1.2 **Development Cycle Implied by Design**

Now that we have defined what is implied by design, we shall now consider the process is implies. Theoretically, design developments are presented as a process consisting of five main **design phases** [11]:

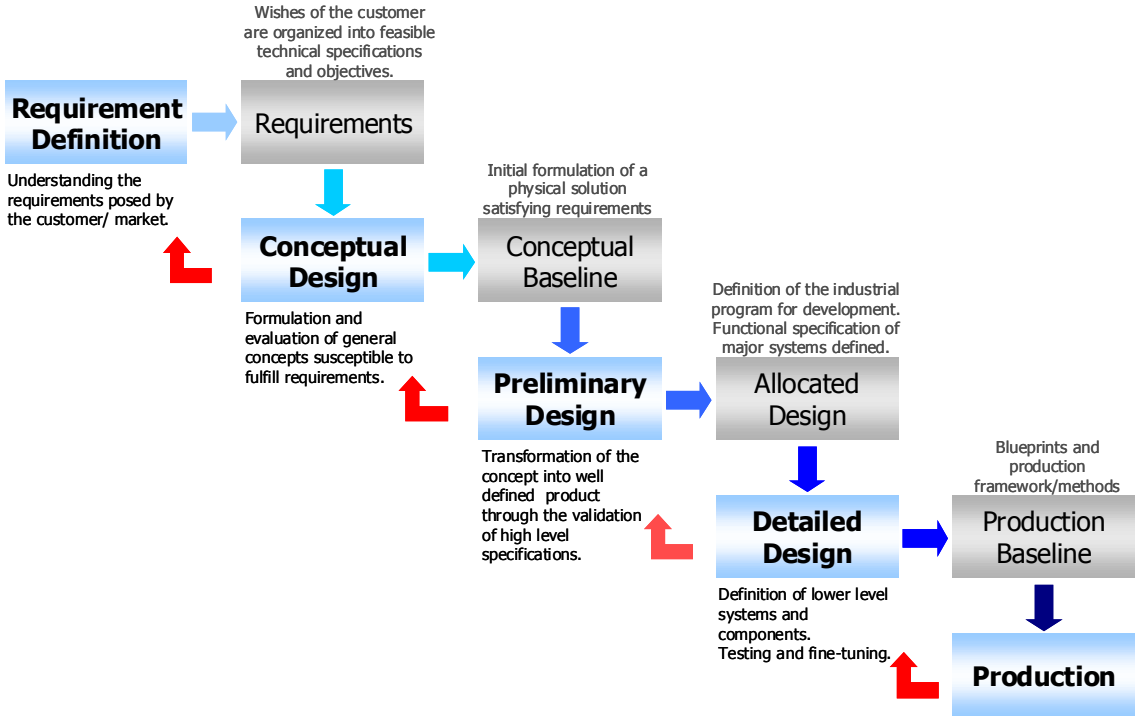


Figure 1: Design phases

The universality of these phases results from the fact that each step will address a different dimension (or level) of the realization of the final solution. The blue arrows represent progression toward a physical solution. The red arrows highlight the feedbacks necessary to the optimization of previous decisions. The blue boxes represent the five phases of the development, and the grey boxes, the intermediate outcomes linking the different phases.

The **requirement definition phase** defines the design problem in a fashion that will guide the definition of the concept. Therefore the main activities in this phase are first to capture the need of the customer and the stakeholders, then to balance those needs to what is technically feasible.

In the **conceptual design phase**, several families of solutions are considered. Each family of solutions, is a mental representation (nothing was constructed yet) of the physical solution. This mental representation is what we refer to as the **design concept**. The key specificity of conceptual design is that a large scope of concepts should be investigated. In a way, conceptual design is about making sure that we are not "drilling next to the gold mine" (i.e. that we are missing the opportunity to pursue better solutions). Therefore, the focus is more on the breadth of the investigation rather than on its depth. The precision of the analysis is only limited by maintaining the ability to compare accurately between alternatives.

Conceptual design is concluded by the selection of one or maybe a few concepts that were recognized as the most promising solutions to the design problem. The selected concepts are then processed in the **preliminary design phase**. Unlike conceptual design, preliminary design is concerned about depth rather than breadth. The investigations are often more focused on specific disciplines, or specific subsystems. The goal of these investigations is the validation of the concept previously defined and to make sure that no design detail will prevent a successful development ("**showstopper**"). The other important activity characterizing the preliminary design phase is the definition of component functional specifications. The **functional specification** corresponds to the design problem for the component. It formulates the focused need within the architecture that will initiate the development of the components.

The development of the components corresponds to the **detailed design phase**. Based on the functional specifications identified in preliminary design, the component designers will create a physical solution. The ultimate objective of this phase is to define

all the details necessary to have a complete description of the finalized system in preparation for its production.

The **production phase** captures several phases of the life cycle of the system. Chronologically, production is initiated by the manufacturing of the components and their physical integration. Production will then include testing and certification activities. This phase prolongs itself in the sustained production to satisfy demand.

The previous description has described the process in its progression forward. In Figure 1, the corresponding flow is the one represented with the blue arrows. The other flow, represented with the red arrows, highlights the fact that information about the activities downstream is needed. If we assume that the blue path is a chronological representation of the design activities, the red arrows indicate that the designer needs information defined in the future. This need is at the root of all design problems. The ability to predict or approximate downstream information is necessary to make informed decisions and streamline the design process.

2.1.3 **Concepts of Concurrent Engineering**

This need to capture latter phase knowledge and information has been the focus of the concurrent engineering community. The initial observation for their argument was the fact that changes in the selected design baselines mostly occur late in the development. These changes are expensive and penalizing, since the later we are in the development process, the more committed we are to a solution (efforts were already dedicated in investigating the solution, a manufacturing setup was prepared, etc…). To illustrate the need to capture knowledge associated with latter phases, the notional comparison was made between two organizations. The first one was a successful automotive company and the second its struggling competitor [12]. This comparison is illustrated by the following figure.

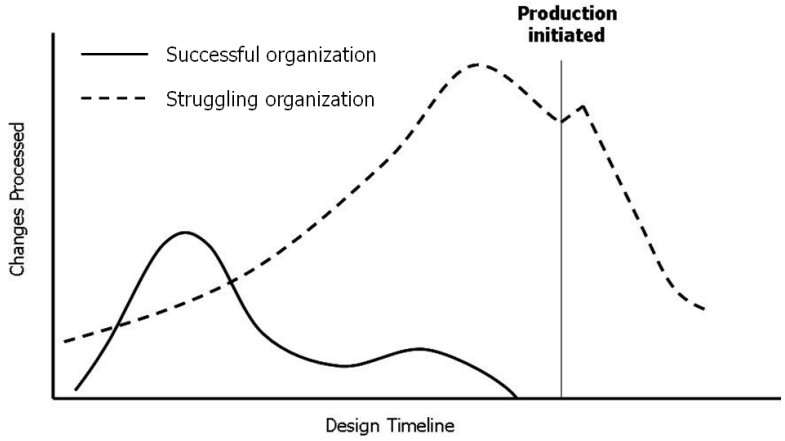


Figure 2: Changes in developments [12]

The solid line corresponds to the successful organization and the dashed line to the struggling company. One can observe that the successful organization will have more instability in earlier phases of the design process, but will stabilize the changes before reaching the production initiation milestone is reached. In the mean time, the struggling organization will experience a comparatively more serene design process in the initial phases, but which will eventually escalate to an "avalanche" of changes in latter phases. The consequences of this avalanche are costly manufacturing reconfigurations and even product recalls to customers (most costly due to the impact on the image of the organization).

The difference between the two situations above was explained by observing the design practices of each organization. In the earlier phases of its development, the successful organization tries more alternatives and performs broader and more flexible investigations. These investigations explain the apparent instability in the early phases of development but build up to a mature baseline in later phases. On the other hand, the other organization performs serial investigations (one discipline is investigated at a time). Given the absence of inter-disciplinary investigations, no compromise is necessary in the earlier phases. As a consequence, this organization will rapidly commit to a baseline. But as the number of disciplines included in the analysis increases, incoherencies and

incompatibilities in the baseline arise and fundamental changes are necessary to achieve coherence and make progress in the development.

To illustrate the situation, we may consider the evolution of "knowledge" and "design freedom" with respect to the chronology of the development. The term "**knowledge**" refers to the degree of understanding of the design problem. For example, the level of knowledge at the beginning of developments is low due to the fact that the designers are not yet aware of which technical challenges must be addressed in order to get to a solution. The level of knowledge then would increase as technical gaps, tradeoffs and potential showstoppers are identified and addressed. Then as different solutions are investigated, the level of knowledge increases since the investigator knows what works and what does not.

The term "**design freedom**" refers to the degree to which decisions made up to that point are going to constrain the designer in adapting the design baseline to newly identified challenges. It is interesting to note that unlike the increase of knowledge which is inherently positive, the decrease in design freedom is more ambiguous. As decisions are made, these decisions will stop design negotiations and will decrease the design freedom. Making decisions is necessary to the completion of the design process. Therefore, a decrease in design freedom can be seen either as a progression toward a solution, or as a decrease in the possibility to adapt the current baseline to arising technical problems.

It is to address that ambiguity that the evolution of design freedom must always be considered in the context of the degree of knowledge. The reduction of design freedom in a context of limited knowledge is a sign that binding and potentially suboptimal decisions have been made, while the same reduction with a higher level of knowledge highlights the fact that decisions were made to address well understood issues. If we now come back to the successful and struggling automotive companies' example,

Figure 3 presents notionally how knowledge and design freedom evolve throughout their design process [11].

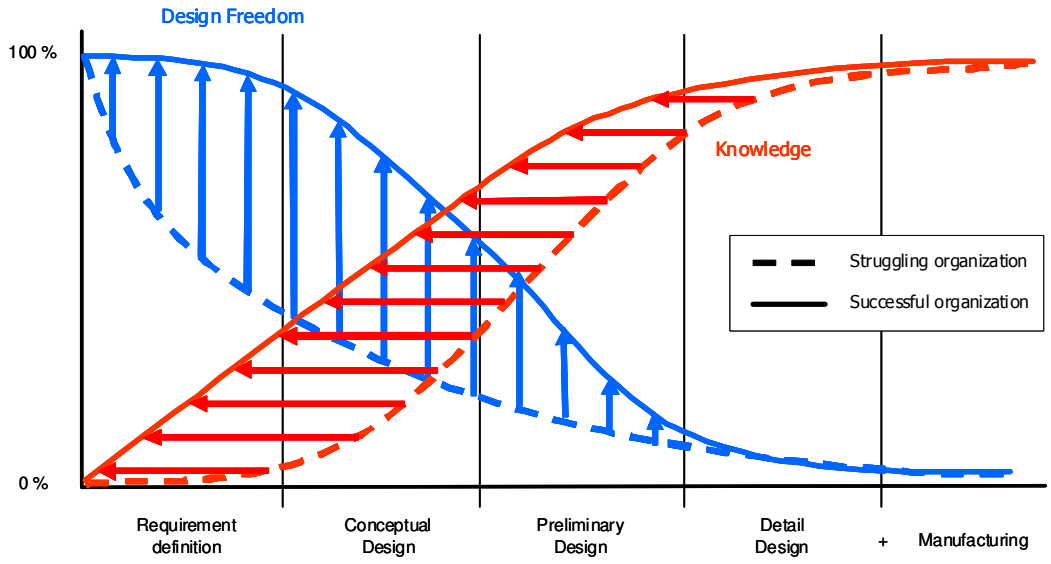


Figure 3: Feeding information forward [11]

We can see that the struggling organization has a steep decrease in design freedom upfront. In the mean time, there is very little evolution in knowledge, which seems to indicate that uninformed decisions were made. On the other hand, we can see that the successful organization is not cutting off design freedom (i.e. is not making binding decisions) but is building up knowledge instead. This approach allows them to make educated choices which occur in later phases at a comparatively higher level of knowledge than for the struggling organization.

Operationally, this difference in strategy was achieved using **concurrent engineering**. Concurrent engineering is a design practice which involves simultaneous consideration of disciplines, early in the design process. This practice is to be contrasted with the serial approach where disciplines are considered one after the other. In initial phases, only a few disciplines are considered. As a result they are not constrained by the disciplines considered in latter phases. Their analysis is artificially simpler since no

negotiation is necessary. But as we proceed in the analysis, decisions become increasingly constrained, and feasible solutions are increasingly difficult to identify.

This situation is illustrated by the following figure representing notional Gantt charts for serial and concurrent engineering developments.

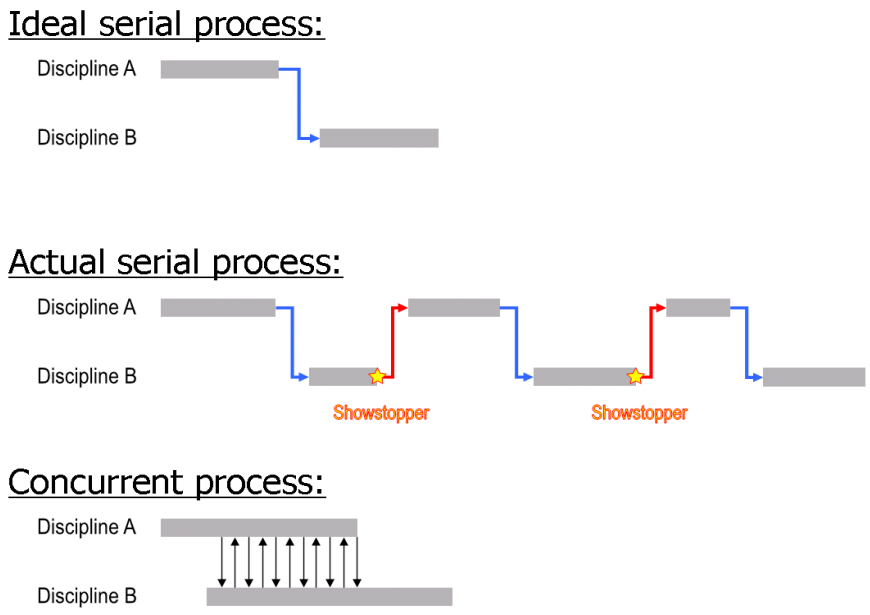


Figure 4: Serial vs. concurrent engineering

In the example represented by Figure 4, disciplines A and B are mutually dependent. In a serial development, discipline A is going to perform its analysis first. Since no investigations in discipline B have been performed yet, the expert in A is going to make assumptions about B. Ideally, those assumptions should be confirmed by discipline B. Based on these assumptions, experts in A are going perform their analysis and freeze their design baseline. By freezing the baseline, the design freedom of experts in B is reduced. If assumptions on discipline B are erroneous, the resulting incompatibility may result in a **showstopper** (i.e. situations where no feasible solution can be identified with the current baseline or where performance is not sufficient). The showstopper implies that previous studies should be reopened. In an industrial development, a showstopper is a chief engineer's worst nightmare. The reason is

illustrated by comparing the “Ideal” to the “Actual” serial processes in Figure 4. The estimated timeline (which corresponds to the ideal timeline) is very different from the iterative process which eventually occurs to address the showstoppers.

Using a concurrent process as described by Figure 3, reduces this risk. If we consider the Gantt chart corresponding to the concurrent process in Figure 4, we see that trades in B are conducted jointly with trades in A. Instead of relying on assumptions, the experts in discipline A exchange information with experts in discipline B. As a result, information can be exchanged and design solutions negotiated between the two contributing disciplines. Since experts in B are directly kept in the loop, the risk of a showstopper due to false assumptions is avoided. This explains why concurrent engineering promotes bringing knowledge forward in the design process; instead of making un-founded assumptions about B, decisions are based on validated knowledge.

On the flip side, concurrent engineering increases the complexity of the study. By integrating multiple disciplines in the analysis upfront, concurrent engineering requires the integration of potentially radically different perspectives. Each perspective will motivate a different solution to address its particular aspect of the problem as illustrated by Miller [13].

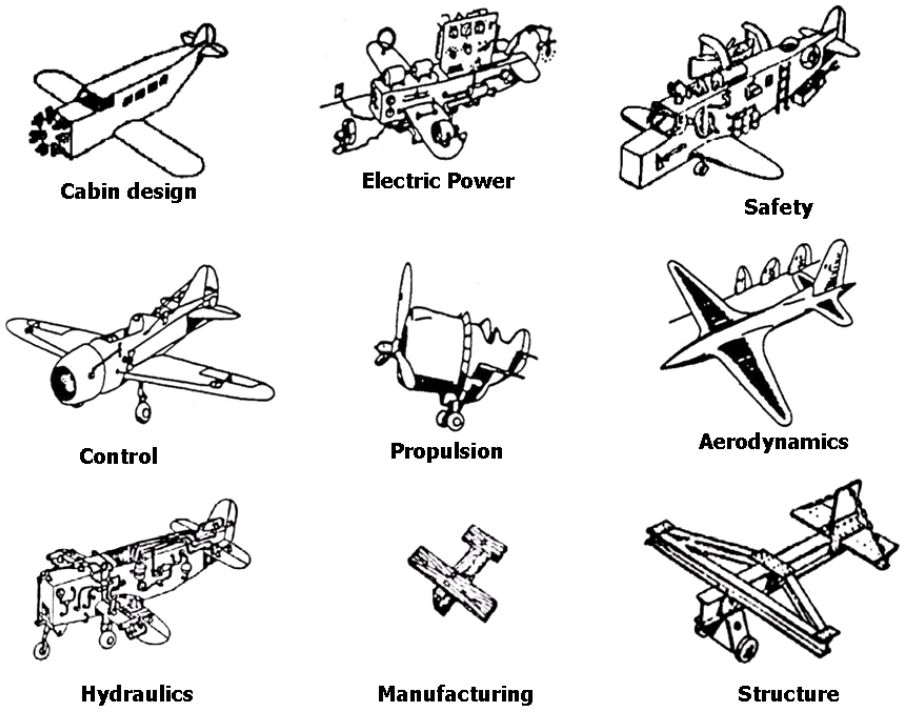


Figure 5: Disciplinary perspectives by C.W. Miller [13]

2.1.4 **Implementation Approaches to Concurrent Engineering**

The reconciliation of different perspectives in an industrial context is by itself a great challenge. As highlighted by the vertical arrows in Figure 4, concurrent engineering requires a great deal of communication to be exchanged between disciplines beyond their difference in perspectives. Different disciplines often use different "languages" in their study, although their focus of study is often overlapping. As a result, the same thing is referred to in different ways. In order to address those disciplinary integration studies, two main families of tools are available. The first family can be referred to as **Multi-Disciplinary Analysis** (MDA) which consists in the aggregation of numerical models pertaining to the different disciplines contributing to the design activity (developed in more detail later in this chapter). The second family is the **Integrated Product Team** (IPT). The IPT is a design team composed of experts from different disciplinary backgrounds working together toward the definition of a physical solution. Both the

MDA and IPT share the objective of integrating the different disciplinary perspectives in order to optimize the aircraft as a whole.

But from a computational standpoint, multi-disciplinary optimization is not a trivial task. As observed earlier, serial approaches were penalized due to the fact that assumptions were made about succeeding disciplinary studies. Even though these assumptions were constraining for later studies, they did however offer an important advantage. The advantage of laying those assumptions was the fact that they were fixed (one picks a value/baseline/setting and then moves on with the study). In a concurrent engineering approach, these formerly fixed assumptions now become a whole new set of design variables that must be explored and optimized. The consequence is that concurrent engineering may result in trades with an overwhelming number of design variables.

In an industrial context driven by deadlines, the higher dimensionalities due to concurrent approaches imply higher risk in the short term design activities. As we saw previously in Figure 3, serial approaches also imply risk, but a risk that will express itself later in the development. Therefore, it is essential to find the proper balance in the level of abstraction as both extremes may result in development risks.

2.2 Design in an Industrial Context

2.2.1 **Challenges due to the Project Scale**

The development of a new aircraft is a colossal industrial undertaking. To illustrate this scale, one may consider the development of the A380. The A380 was a twelve billion Euro (16.4 billion dollars) development project [14] which involved up to 30,000 individuals across 4000 collaborating organizations [15], [16]. Given the magnitude of the investment at stake, and the coordination problem in such a development, having the ability to carefully monitor and control the development process has paramount importance.

In this development, the phases that are perceived as the most complex from an industrial stand-point are preliminary and detailed design phases. During these phases, the design activity is peaking in terms of number of people and organizations involved in the design process. The development focuses on the lower level design tasks ("nuts-and-bolts") where the volume of design tasks is large, but the interconnections between tasks are weak. In order to facilitate, and monitor those different component-level developments, several methods and tools were developed. The result is the many "document based" system design methods. The "document based" method is a monitoring and coordination process that focuses on the interfaces between design tasks, by putting in place reporting and communication processes which describe:

- The latest baseline of the part/component being designed
- The interfaces between elements currently designed by different groups of people
- The progress in the design task based on milestones

Several tools like Doors or CATIA have proven their ability to support these processes. Doors provides a means to monitor progress, and manage documentation [17].

Catia/Delmia from Dassault Systems is an example of a tool that can be used to monitor physical baseline and manufacturing methods.

These processes are now well integrated into the design habits within large aerospace organizations. The customer base of tools supporting these processes illustrates that fact. For example, Door's customer base includes Lockheed Martin, Northrop Grumman, Boeing, Bombardier and EADS [18].

2.2.2 **A culture of clustered-concurrent design**

2.2.2.1 Definition of Clustered Concurrent Engineering Developments

The tools and methods described previously allow the deployment of a compromised strategy between concurrent engineering and serial design. In this thesis, I will refer this strategy as a **Clustered-Concurrent Engineering** development.

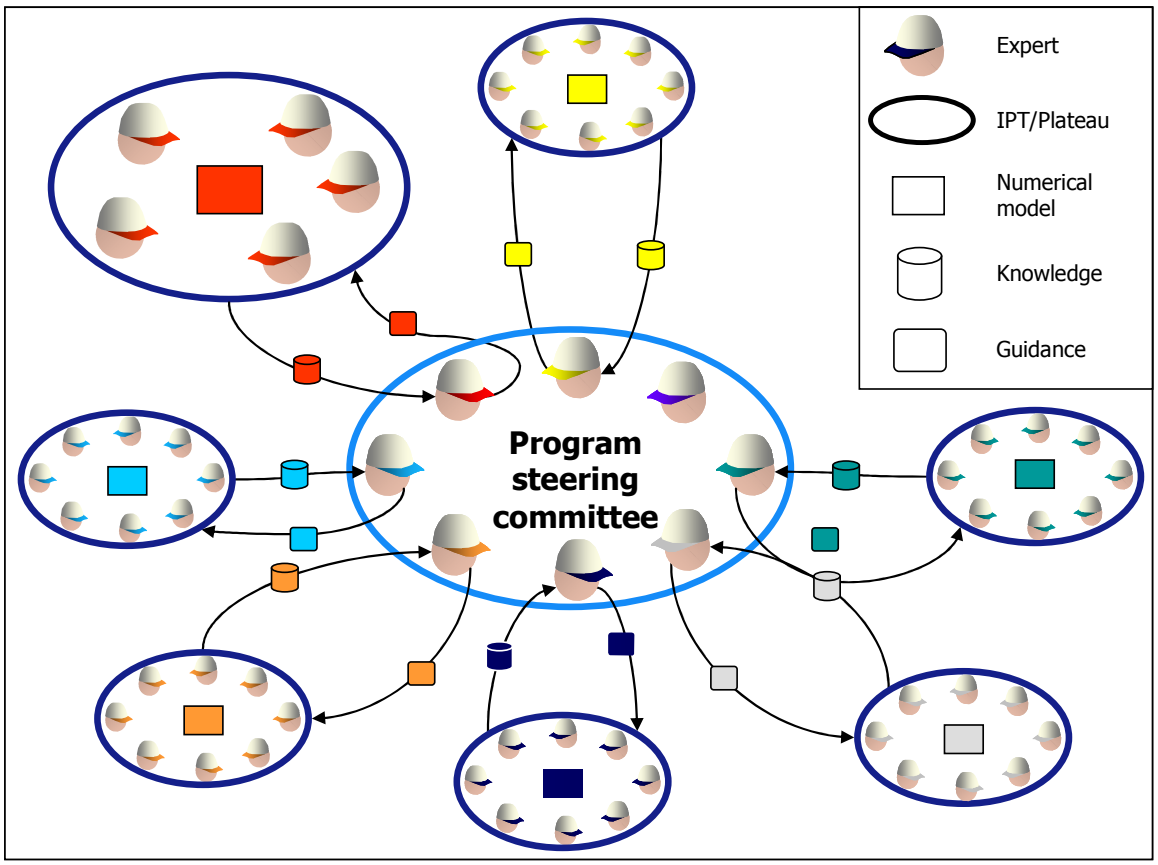


Figure 6: Clustered-concurrent development framework

The design activities can be dispatched over a constellation of IPTs (referred to as **design plateau**). The plateaus (represented in dark blue in Figure 6) are IPTs dedicated to local and specific design perimeters of the aircraft within which the coupling between design problems necessitates the application of concurrent developments. An example of a design plateau is the powerplant plateau which coordinates the development of the nacelle, the engine and all systems installed in the nacelle. The plateau design decisions are integrated and monitored at the aircraft program level by a program steering committee using the system engineering process discussed earlier. The design process implied by this development structure is described in Figure 7.

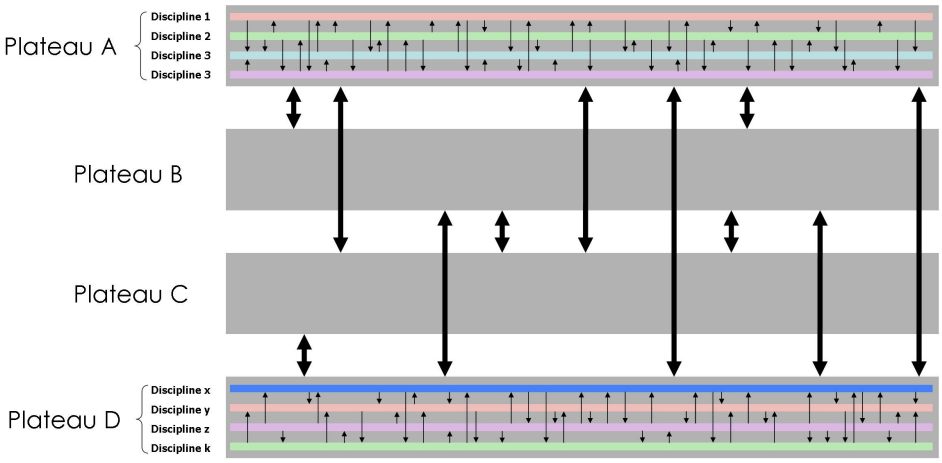


Figure 7: Clustered-concurrent engineering approach

Figure 7 presents a notional Gantt chart of the design activities during detailed design developments. The horizontal dimension represents the timeline. Each grey box represents the design activities of a different plateau. Within each plateau, trades pertaining to its perimeter are conducted concurrently. The intense exchange of information internal to the plateau is represented by the small arrows joining the colored bars. The colored bars represent the activity of the disciplines represented in the plateau. Between plateaus, the exchange of information is more sporadic and addresses the integration of the parallel design efforts. The inter-plateaus exchange of information is carefully managed by the system engineering process which facilitates it but also

regulates it. It is important to note that this approach provides the means to perform trades both in a parallel fashion and also concurrently at a local level. The design activities within each plateau are relatively decoupled and allow design activities to be performed simultaneously.

2.2.2.2 The Essential Role of Clustered Concurrent Engineering

This parallel approach allows the application of concurrent engineering methods, while facilitating the industrial process constrained by its scale. But the emergence of parallel developments is not only motivated by concurrent engineering concerns. The multinational aspect of Airbus has reinforced this situation. The fact that different plateaus are located in various geographical sites has reinforced the need to compartmentalize the studies. Similar observations could be made about BCA. The multinational development of the B787 elements (wings in Japan, engines in the United Kingdom, etc…) provides another motivation for concurrent clustered engineering.

With the strategy of increased risk and cost-sharing (as evoked in Chapter 1) the airframer delegates larger subsystem developments to its technological partners. In this context, the detailed design developments are on the verge of even a greater degree of segregation. Even-if the developments are done in a spirit of inter-organizational collaboration, the fact that the technological partner will also potentially work with a competitor imposes limitations on the exchange of design information. This need to protect intellectual property accentuates the segregation between design efforts.

2.2.2.3 Clustered Concurrent Engineering Role in Architectural Conservatism

From a detail design practice, the clustered-concurrent design approach has also become part of the organizational culture of the company. If this culture serves the development needs during the preliminary and detail phases, it also has a tendency to become an obstacle to conceptual trades. In order to benefit from plateau-based developments, the design problem must be broken down into weakly interdependent

design problems at subsystem level. In the early design phases, the design problem is only defined at the aircraft level (number of passenger, operational range, etc…). The subsystem (or plateau-level) design problems are not defined since the architecture concept is not formulated at that point. It is difficult to decouple the analysis at the subsystem level, as it depends on the aircraft system architecture, and the interaction between the subsystem components will be different. Performing conceptual trades with a CCE approach forces a breakdown of the aircraft design problem. This role applied to the central IPT represented in clear blue in Figure 6 can not be performed using the same tools and methods as those used in detailed design (where the focus is on monitoring interfaces and steering developments). Hence, the breakdown tends to be defaulted to the one corresponding to previous architecture. This approach enables the recycling and application of lower-level tools, methods and lessons learned from previous programs. But it also eliminates the ability to considered new architectures.

Therefore if a clustered-concurrent approach is used in the conceptual design phase, the necessity of a breakdown and subsystem level objectives will maintain the trades within an existing architecture. From this observation we can conclude that the problem observed in chapter 1 is, in fact, a consequence of the lack of ability to perform conceptual architecture trades in a clustered concurrent engineering approach.

2.3 Challenges in Applying Concurrent Engineering Approaches to Architecture Conceptual Development

In the previous section, a focused problem was identified in the clustered-concurrent approach employed by the industry in its developments. This focused problem is located at the center of the development framework represented in Figure 6. In order to formulate the constraints and needs implied by this problem, this section will provide a more detailed analysis of the objectives, the activities and the actors of conceptual design. This analysis will then be followed by a description of the challenges related to conceptual architecture developments.

2.3.1 **Overview of Conceptual Activities**

The conceptual design phase has a great strategic importance in an industrial development. As we observed earlier, the phases after conceptual design have the tendency to be overwhelmed by the detailed trades necessary to the final development of an aircraft. As a result, the conceptual design phase offers the one and only occasion in the program to seize the opportunities offered by aircraft level trades.

In the early phases of development, the aircraft level requirements are starting to emerge. Based on these requirements, the objectives are:

- To identify the best aircraft concept possible
- To formulate the best design framework necessary for detailed design activities

The identification of the best aircraft concept possible requires exploring alternatives, which implies analyzing and comparing competing concepts. The analysis consists in determining the expected aircraft attributes for each competing concept. These attributes are defined based on the aircraft level requirements. These requirements impose functional specifications on the subsystems constituting the aircraft. Deducing the basic subsystem attributes from their respective functional specification is referred to as the

sizing of subsystems. In a second step, the aircraft level attributes can be defined based on subsystem-level attributes. This activity is referred to as **synthesis**. Therefore, the analysis of the competing concepts can be broken down into two activities: sizing and synthesis.

But beyond the analysis, competing concepts must be compared. This comparison defines the level of depth necessary for the analysis. In a sense, the analysis should be only as deep as is necessary to distinguish between the alternatives. But this statement may be quite misleading as it does not imply that only shallow analysis is necessary.

A common claim by aircraft system architects states the "the devil is in the details". Often the most critical trade-offs are decided on differences which could almost be deemed as insignificant. The following statement from Boeing 787 chief engineer, Michael K. Sinnett, provides a motivation for careful conceptual design analysis.

> "When we decided on electric pressurization, it lowered the aircraft weight by 1000-2000 lb […] but the numbers got muddied as the 787 got integrated. It is hard to say where the weight has gone."[19]

The empty weight of the Boeing 787-8 is about 252,000 pounds. The statement above indicates that a decision was partially made based on a difference which accounts for less than one percent. We can also see that the conceptual difference was affected by the system integration aspects. Therefore, we can observe that the importance of conceptual analyses does not only reside in estimated quantitative conclusions, but also in an engineering learning process which enables capturing of the margin for improvements (technological opportunities), and programmatic risks.

The complexity of the integration problem can be traced to the complexity of the mission itself. With requirements as varied as vehicle performance, structural integrity, or passenger safety and comfort, the solution is bound to be complex and integrated. This complexity results from the extreme technical challenges implied by performance requirements. It will be integrated because there are no simple solutions to a design

problem where the outcome must have the capability to take off on a field that is less than 3600 ft, fly at mach 0.8 at 35,000 ft and at the same time, brew coffee for 300 passengers.

The fact that the mission requirements necessitate an integrated solution implies that the aircraft will be constituted of multiple elements which will have to work together. This ensemble of elements is what constitutes the **Aircraft System Architecture**. The elements constituting the architecture are referred to as **subsystems**. Each subsystem will require a specific form of design expertise. This type of expertise is not necessarily disciplinary per se as it refers to the ensemble of knowledge necessary to size a specific type of subsystem solution. This knowledge will therefore correspond to things like turbomachinery design, electric system design, cooling air cycle machine design, hydraulic design and so forth. The design experts are dedicated to the sizing aspect of the analysis

In addition to the design experts, another type of expertise is necessary which corresponds to the disciplinary experts. Given the tight integration of subsystems, relating subsystem-level to aircraft–level attributes and requirements is not trivial. The analysis performing the synthesis necessary to make this link is performed by disciplinary experts. Disciplinary experts are aerodynamicists, stability and control experts, or aircraft performance experts. Their role is to aggregate the attributes of the subsystems and derive aircraft level attributes. The role of the disciplinary experts is to support the synthesis effort necessary to the evaluation of the value of the aircraft concept or to specify requirements for subsystems.

Hence, the definition and analysis of architecture concept alternatives is never a single person's job. Although these experts are focusing on a part of the design problem, they all contribute to solving same global problem. The sub-problems are bound to influence each other (the engine design influences the electric generator design and the actuator' will influence the hydraulic pump', etc…). Because of these influences, multi-disciplinary/expert trade-offs are necessary to find the appropriate solutions to the global

problem. As a consequence, although each expert takes part in the decision making process by making decisions at their level and within their perimeter, their decisions are federated by a single global objective at the aircraft level.

2.3.2 **The Role of the Architect**

In order to achieve this agility and depth in the analysis, the notion of a close exchange between disciplines, as prescribed by concurrent engineering, becomes critical. In order to achieve these objectives, it is necessary to have an appropriate orchestration of the disciplinary efforts and facilitate the necessary negotiations between disciplinary perspectives. These concerns are at the heart of the architect's role.

The initial role of the **architect** is the formulation of requirements. In the building construction industry, the architect is the person interfacing with the customer. Together with the customer, he defines the needs. From these needs, the architect will define the style and type of house: Is it going to be a one or two story house? Is it going to be a skyscraper or a long building? For the most complex projects, this architect will be working in collaboration with disciplinary experts. A fascinating example is provided in reference [20]. In the development of his design proposal Mr. Piano and Okabe (architects for RPBW Japan) have performed an exemplary conceptual study. Customer requirements and disciplinary aspects were considered concurrently. The definition of the building was driven by several simple but conflicting objectives. For example, the main terminal had to be spacious, but quiet, and with a uniform climate distribution. These conflicting requirements were addressed by the concurrent development of a ceiling concept, between an aerodynamics and a structural expert. The final concept was high ceiling shaped to provide a uniform distribution of the air conditioning flow over the entire terminal. The solution allowed the use of large low speed nozzles quieter than smaller distributed vents, and the shape of the ceiling provided a uniform draft-free

distribution of the conditioned air in the terminal. A representation of the main terminal and its iconic ceiling is provided in Figure 8.

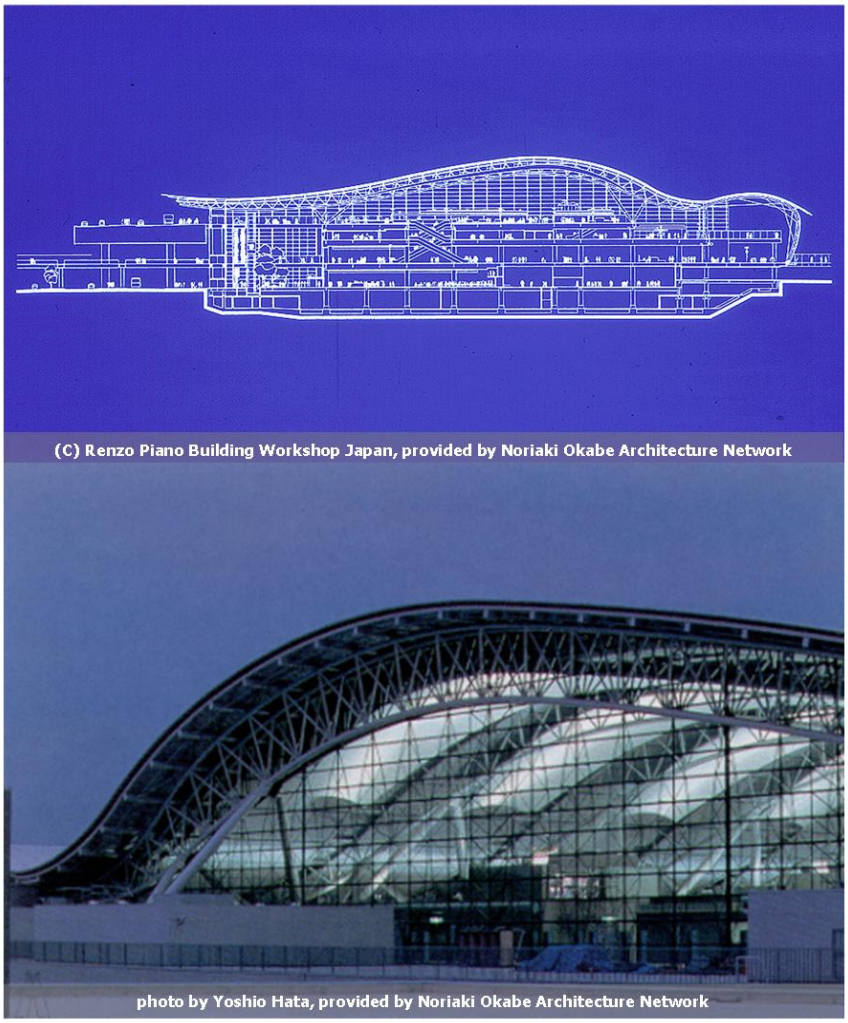


Figure 8: Curves and structures of the Osaka Airport

This role definition is not only for a building/construction architect. The term architecture refers to the "underling structure of things" [21]. Whether it is a building or an aircraft the fundamentals are the same. The role of the architect is:

- To capture the need motivating the design at architecture level
- To initialize the formulation of a solution under the form of architectural concepts.
- To facilitate the integration of knowledge supporting the development of the architecture.

It is also important to note that the experts contributing to initial trades are also in a position of architects. These experts must capture the environment in which their disciplinary analysis or subsystem will be integrated. By doing so, experts are also capturing the need motivating their own design sub-problem. The fact that they propose solutions at their sub-problem level places them in an architecting position within their perimeter of expertise. Finally, the knowledge that the expert is contributing to the conceptual trades is often the result of another level of analysis, which itself relies on the integration of knowledge at a lower (component) level.

2.3.3 **Integration of Knowledge**

In order to integrate new technologies that can not be integrated in traditional architecture, it is necessary to perform what is referred to as a **bottom-up development**. The bottom-up approach consists of integrating information about the lower component level then consolidating it up to the aircraft level. This approach is in contrast to the traditional **top-down development** based on the framework offered by conventional architectures. The top-down approach uses the top-level aircraft requirements to initiate the subsystem level trades. But in order to be able to use this approach, relationships between aircraft level requirements and subsystem level parameters are necessary. In a new architecture context these relationships are unknown. Therefore if innovative technologies are to be considered in conceptual design developments, the need to integrate design expertise is unavoidable.

The integration of disciplinary knowledge goes well beyond the leadership role of an architect. Their integration within a concurrent process requires that information is exchanged between contributors. The objective of this section is to describe some key aspects of this exchange and understand some of the group dynamics which tend to arise. In order to structure this description, this section will be based on the comparison of two approaches to concurrent engineering. The first organizes the exchange through human

interactions; the second is based on the integration of numerical models representing the human expertise.

The numerical approach is based on Multi-Disciplinary Analysis (MDA), which is a field of research supported by both the industrial and academic worlds. The MDA field investigates methods to efficiently relate, within a numerical platform, the analyses traditionally conducted by different disciplines. This requires that the various contributing experts formulate their knowledge into numerical models that can interface with other disciplines. Recently, several research efforts were carried out within Airbus [22], EADS and the European aerospace community at large (VIVACE, and Crescendo) [11] to develop their capability in numerical tools to support developments. On the academic side, one of the most active research fields in the design community is MDO (Multi-Disciplinary Optimization), which investigates new design applications enabled by the deployment of MDA. A graphical representation of the MDA is provided in Figure 9.

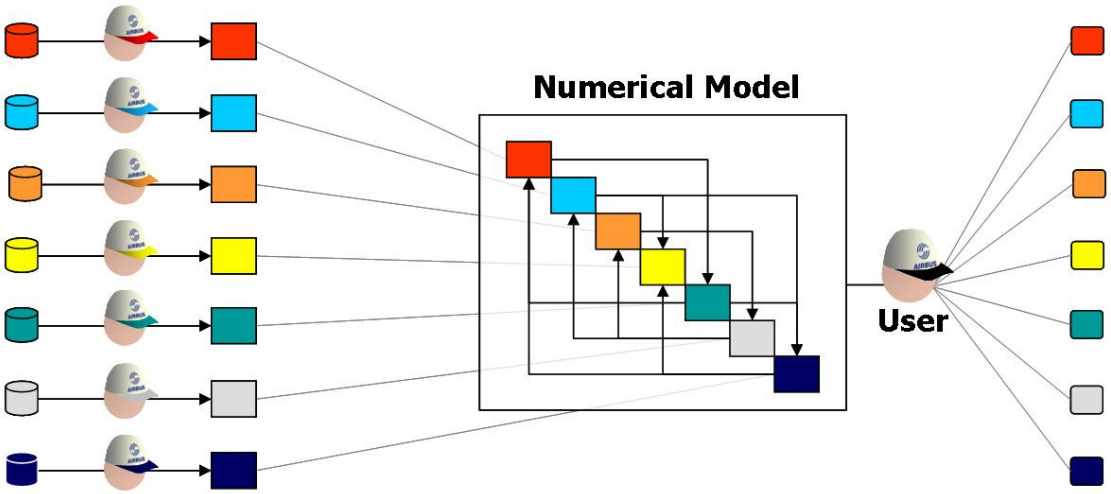


Figure 9: MDA based approach

On the other hand, the IPT based trades are based on a group of experts working together. The lines of communication are supported by human exchange (discussion, taskforce, workshops, meetings, brain storming sessions, etc…) or by punctual exchange

of information in a textual or numerical form (email, presentation slides, data sheet, etc…).

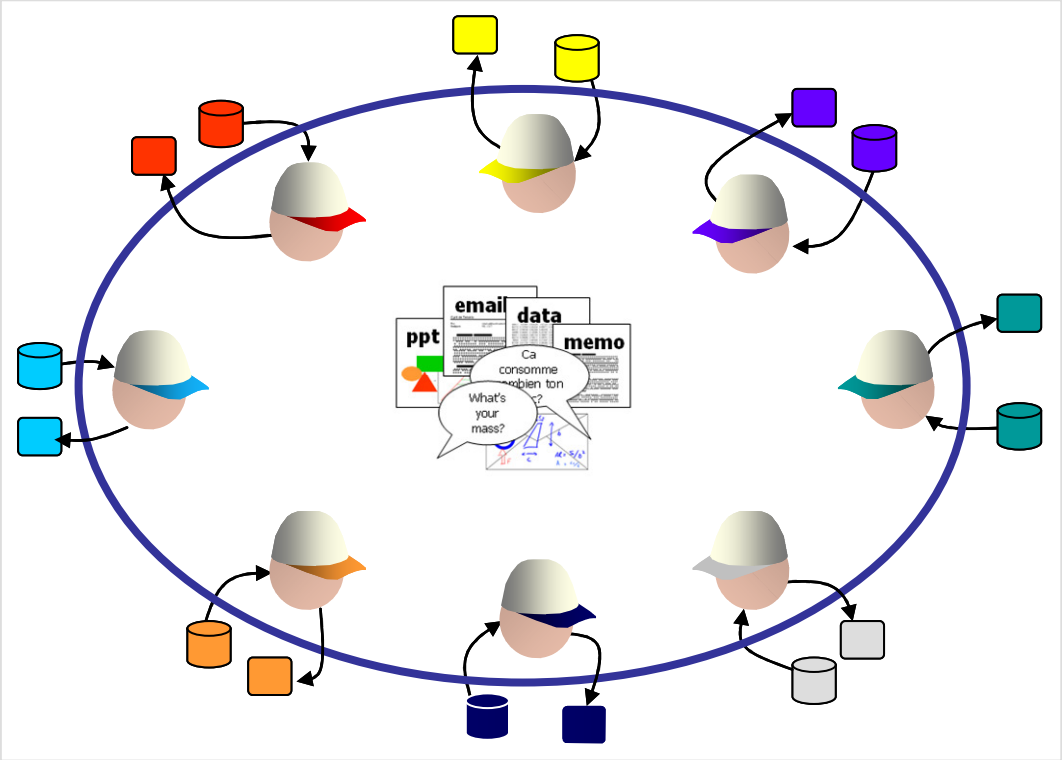


Figure 10: IPT based approach

This section will contrast the IPT and MDA approaches as if they were incompatible. This incompatibility is not a necessity. But segregating and opposing these approaches highlights their mutual strengths and weaknesses.

2.3.3.1 Communication between Experts:

Different experts will have different backgrounds in their professional experience and training. As a result, they are likely to use a different vocabulary to describe things, and to see the design from a different perspective. The difference in vocabulary can be illustrated by the experts in turbine design who refer to their turbine elements as buckets (we can recognise their legacy from river mill designers) while the compressor expert would refer to them as blades. As a result, one of the two most important contributions of MDA is the fact that it forces the experts to unambiguously identify their

interdependencies and agree on the correspondence of the different terms contributing to their trades.

2.3.3.2 Promptness of Execution

The most important advantage of MDA is the automated exchange of information between the contributing analyses. Each of the contributing modules corresponds to the analysis of one expert. The MDA automates the exchange of data between the contributing analyses by specifying upfront the linking variables. Based on this construct, the overall analysis can be considered as being a monolithic model on which traditional design techniques for optimization, and decision making can be applied.

On the other hand, the line of exchange of information is not automated for an IPT. The information is requested, and provided through significantly slower forms. These forms include oral communication (direct or over the phone), textual document (memorandum, report, presentation or email), and data sheets. These practices impose significant overhead on the analysis activities of the IPT participants. This aspect imposes a dire limitation on the number of trades that can be performed in a study and prevents the integration of many design methods which would be quite helpful in improving the outcome of conceptual design activities.

The promptness of execution of MDA and its apparent simplicity once set up has fostered the development of many methods which were proposed in the academic field. Some of these methods will be reviewed in Chapter 3 of this document. But these important strengths of the MDA approach also have the tendency to hide some of its limitations, which still prohibits its integration in some industrial applications.

2.3.3.3 Complexity of the Analysis

The IPT approach can be considered as the “manual” or “custom-made” approach to architecture analysis. Each aspect of the architecture will be analyzed with the appropriate tool, database, or knowledge relevant to the problem and its circumstances.

The choice of analysis will be defined by the expert based on his/her consideration of the context. For example if an electric generator must be sized. Whether it is to be integrated in a waterproof zone or in a zone exposed to humidity, the sizing rules applied will be different. This necessary adaptation can easily be considered by a human expert.

In a similar fashion, the interfaces between the different fields of expertise will also depend on the architectural concept in which the systems are to be integrated. Depending on the physical and functional interfaces present in the architecture, the analysis will require different elements of information. For example, we may consider the same generator as above. If the generator is receiving its mechanical energy from a turbofan gear box, or from a hydraulic motor, the elements of information that will need to be exchanged will be different in each case. If connected to the turbofan, this architecture will imply a design relationship between the engine expert and the generator expert, in the other case, the design relationship will be with the hydraulic expert. The had-hoc approach implied by the IPT allows the expert to identify the necessary interfaces and to focus only on those deemed as necessary to the analysis. For both the selection of the appropriate model and the exchange of information, the experts constituting the IPT can customize their approach to maximize the fidelity of their analysis while minimizing the amount of exchange. The disadvantage of the IPT is its "manual" means of exchanging information. Even if the exchange of information is minimized, the transmission of the necessary information between experts is time consuming.

On the other hand, the MDA setup implies an automated and systematic approach. An ensemble of basic logical rules tunes and adapts the analysis so that it may represent the specific concept alternative under consideration. Numerical analyses are good at capturing a continuous variation of parameters which does not change the fundamental structure of the logic on which it is constructed. In most cases, architectural alternatives

can not be represented by a continuous variation of parameters. Different alternatives will require fundamentally different structures in the MDA.

In the example of the water-free and wetted environment for the integration of the generator, we assume the sizing rules for each situation are different. Either the MDA will require both sizing logics with a switch somewhere that will trigger the appropriate sizing model, or the MDA will be limited in its scope and validity. In the same fashion, depending on the relationships between the elements within the architecture, the links between the different element analyses will be different. On the turbofan driven generator, the generator analysis model will need to be linked with the turbofan model; on the hydraulic motor driven generator, a different link will be necessary. As multiple architecture concepts are considered, the logic necessary to the definition of all the necessary switches can quickly become unfathomable. Therefore, it is difficult to create an MDA environment which can capture architectural alternatives.

In the context of conceptual exploration, complex analyses must be performed recursively to analyze different concepts. Being able to apply previous tools, conclusions and experiences contributes to accelerating the investigation process. An MDA can perform automated analysis on any concept within its space of validity. This analysis does not require significant additional time to perform new analysis while the IPT will have to conduct a new time consuming study.

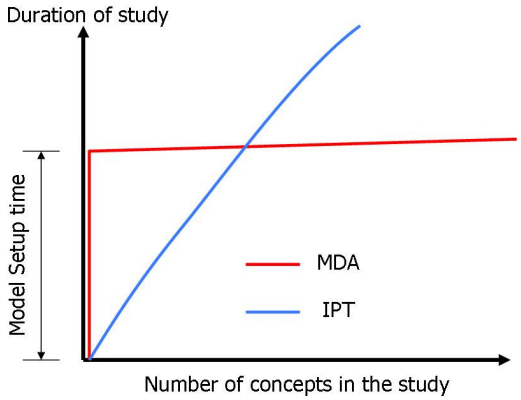


Figure 11: Time associated with performance of integrated analysis

The complexity of the analysis requires a significant investment in effort. In Figure 11, the initial investment is represented by the step in "time necessary". But beyond this initial step, the time associated with every new evaluation within the modelling scope is negligible. On the other hand, the IPT will not require investing significant effort in the setup of the study, but the turnaround in the investigation is significantly slower.

From Figure 11, two observations can be made:

- When there is an investment there is risk. In this context, the risk is that either the model does not work, or that the model it too limited in scope. Given the complexity of the analysis, these concerns can be legitimate and will prevent the development of numerical methods.
- We can consider that the cost associated with IPT investigation is proportional to the number of concepts studied. Therefore IPT based studies will tend to be limited in their conceptual scope. On the other hand for MDA, the weak coupling between cost and scale of the study allow a larger scope of investigation.

2.3.3.4 Estimation of Subsystem Optimization Potential in Architectural Sizing

Sizing is an important aspect of conceptual design. In a bottom-up architecture sizing approach, the evaluation of the overall attributes of the architecture are based on the sizing of its individual subsystems. The sizing of the subsystem is an attempt to predict what the subsystem is going to be like, given its role in the architecture and the technology available to its development. In other words, we can say that sizing is in fact a prediction of what the subsystem design process is going to produce. But, in the same fashion as you have trade-offs at the architecture level (i.e. which architecture concept is the "best"?), the subsystem level also implies trade-offs and compromises. Therefore, the question in the sizing of a subsystem becomes: What is the "best" subsystem going to look like? For example, if an electric generator must produce 100 kVA, it is likely that multiple electric generator alternatives can be considered for this job. Each will have its

own qualities (more efficient, lighter, cheap, etc…), but not all will be equivalently suitable to the architecture

In an IPT investigation, each expert is going to use his/her tacit knowledge to size the subsystem: "Based on my experience, what is the most appropriate solution? Which attribute of my subsystem is known to help the architecture the most (efficiency, weight or reliability)?" Based on the answer to these questions, the most appropriate subsystem solution is proposed. In other words, the expert either knows or defines what the most appropriate solution for his/her subsystem is and will submit it as a sizing estimate. But his/her estimation does not always correspond to the optimal choice. As a result, sizing is either inaccurate (assuming that detailed design would eventually identify the optimal value) or suboptimal.

Identifying the optimal subsystem for the architecture is a difficult task. If we consider ten subsystems and assume that each subsystem can take five different forms, the number of possible combinations is 5^{10} (almost 10 million). Given the computational limitation pertaining to the manual exchange of information within an IPT, exploring the subsystem trades is, generally, limited to what the experts believe is best for the architecture.

On the other hand, even though the MDA is faster, in most cases, it will not be able to capture this level of nuances. The tacit knowledge necessary to capture the possible alternatives and figure out the best alternative is often considered as too difficult to set up in an automated way and is often disregarded. Therefore, most numerical sizing models are based on a simple regression on what the subsystem is supposed to do. In the electric generator example, given the fact that the generator will be producing x kVA, the model will link to x, a value y for the weight, hence abstracting out the subsystem trades and their effect on architecture sizing accuracy and optimization.

From this perspective, neither the IPT nor the MDA offer a satisfactory solution. As a result, when a new architecture is compared to an old one but with optimized

subsystems, the comparison leads to conservative conclusions. Dr. Faleiro leader of the Power Optimized Aircraft program stated in a plenary speech at the 2006 SAE conference [23]:

> We are beginning to realize that when [new technologies] are put together "bottom-up", we have an over-designed, over-weight, under-optimized whole aircraft.

Integrating the effect of subsystem trade-offs is necessary to the sizing of architectures. Not considering them properly implies that the sizing of the architecture will be inaccurate or under-optimized. The traditional architectures, from their past developments, have benefited from subsystem level trades. New architectures have not. Then the comparison of the old and the new compares an optimized architecture to an under-optimized one, hence leading to conservativeness and opposition to changes.

2.3.3.5 Tracing Assumptions

In the early phases of a complex system design, assumptions are necessary to simplify the analysis into a manageable engineering exercise. Regardless of the approach (IPT or MDA) assumptions are used. In the context of architecture analysis, assumptions may apply to different things:

- An interaction between subsystems that might be neglected (ignoring changes in the interaction).
- A limitation in the applicability of conclusions for a model.
- Uncertainty in a factor influencing the results of the analysis (technology factor, mission requirements, etc…)

For the sake of the comparison, let us classify assumptions under two categories.

To define these two categories we consider the following analysis:

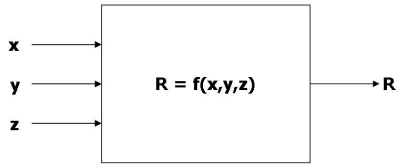


Figure 12: Illustration of assumptions in the context of modeling (part 1)

We assume that as formulated above, the analysis is exact. In most cases, the exact formulation is either impossible to define mathematically or impractical in its implementation. Therefore, the analysis performed (by a numerical model or by an expert) will always rely on a simplified model. In the simple analysis above, the exact model is represented by f(x,y,z). For example, this exact model would correspond to the Navier-Stokes equation in an aerodynamic analysis. In this case a simplified model would correspond to the Bernoulli's equation.

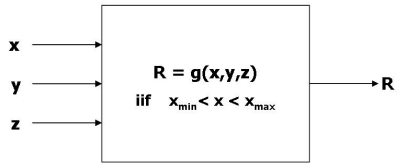


Figure 13: Illustration of assumptions in the context of modeling (part 2)

But all simplifications imply some limitation in the applicability of the conclusions produced by the model. For example, the Bernoulli's equation is limited to flows where viscous and compressibility effects are negligible. This type of simplification within the model corresponds to a first type of assumptions. When these assumptions are taken in an IPT context, the expert will be both making these assumptions and drawing conclusion from the resulting analysis. He/she will be aware of the range of applicability of the conclusions. On the other hand, when an expert delivers a numerical model to be integrated in an MDA, the expert delivers an analysis with its imbedded assumptions. The user of the final MDA may be a different person than the expert who created the model. So the awareness of these assumptions may be lost. Since

the final users of the MDA and the initial authors of the contributing analyses are often two different groups, the validity or the usage of the results produced by the MDA can be compromised. In an IPT this situation is less likely to occur since the person delivering the analysis is the same as the one who made the assumptions.

The second type of assumption is a consequence of uncertainty. Let us consider the following simple model, which links an input variable (x) to a response (R).

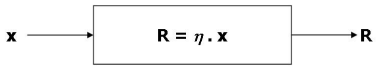


Figure 14: Representation of analysis under uncertainty

Let us consider that x in the model above is the mechanical input of a hydraulic pump, η is its efficiency and R the hydraulic power that it produces. In order to evaluate the power output, both x and η must be known. In the early design phases, many parameters are unknown. Scientific and technological developments are in process, therefore, the estimated efficiency ($\hat{\eta}$) is likely to evolve during the development cycle. In addition to this uncertainty at the heart of the model, there is uncertainty coming from factors external to the model perimeter. In the example of the pump, the mechanical power input may depend on some uncertain factors within the gearbox to which the pump is attached. Several methods can be used to consider uncertain effects in the analysis (Monte-Carlo analysis, robust design methods …). These methods, which will be described in more detail in Chapter 3, translate probabilistic distributions in the inputs into probabilistic distributions in the outputs. But these methods require recursive analysis. Given the fact that the duration and cost of IPT studies is directly proportional to the number of cases investigated, the deployment of recursive analysis is impossible without the automated support of MDA.

As in the absence of probabilistic methods, “design margins” are used in IPT studies. The design margins account for the inherent uncertainty in the design analysis. But margins are often times defined arbitrarily. Given the amplitude of the uncertainty in

early design phases, a wide range of margin values can be justified. As a result, the degree of subjectivity may outweigh the technical study of the problem.

2.3.3.6 Human Factors

As pointed out earlier, conceptual design is a strategically key phase of the aircraft development. Decisions are made about the detail design framework. They will specify how the later developments are going to be organized, how much funding is going to be attributed to different aspects of the development and how each is going to be benchmarked.

As a design concept is analyzed, the expert must estimate the expected attributes of the solution. This estimation is based on the technical potential of his/her organization and its possible technological partners. For example, when Airbus performs conceptual design trades, the electric power system expert will be providing an estimate of the key attributes of the electrical distribution system. This estimation will be based on the technological capability of its department along with those of its partners (Thales, United Technologies, etc…). If the concept is selected, this estimation will become a target for future developments. Therefore, the conclusion of the expert involved in conceptual trades contributes to the definition of his/her future target (or the work of his/her colleagues). Optimistic estimations will impose tough challenges in reaching future targets. On the other extreme, pessimistic estimations are easier to realize but tend to promote the architectural status quo.

When trades are performed by an IPT, the spectrum of possible estimation (pessimistic to optimistic) has been addressed by a technical negotiation process. In this process, the experts initiate the analysis with prudent conservativeness. As the uncertainty about the context of integration of their subsystem decreases, these margins decrease along with the experts' increasing confidence in the concept. During an interview with J.C. Guyot [24], he indicated that the process of "designing an architecture

is mostly about negotiating design margin with experts". Initially uncertainty is high; the experts contributing to the analysis will place artificial penalties on their estimation to account for things that were not analyzed. As the study progresses, the level of knowledge about the concept increases and the uncertainty decreases. In this context the role of the system architect is to make sure that margins are squeezed accordingly.

If an analysis is performed by an MDA, the expert contributions are captured during the setup phase. In the industry, there is a perception that this approach forces the experts to rigidly package their knowledge, where the assumptions (including margins) are defined upfront. Therefore, it is perceived that the setup of the MDA requires their input upfront, but excludes them from directly participating in the trades. To conclude on assumptions, the IPT has the advantage of keeping a closer proximity between the decision maker and the person making the assumptions. Therefore, an IPT is less likely to draw ill-founded conclusions due to an unknown or incoherent assumption. On the other hand, MDA can support analyses which decrease the arbitrary treatment of uncertainty.

2.3.3.7 Conclusions

The IPT has a better ability to explore new architecture concepts because of its flexibility in adapting its analysis. On the other hand, the difficulty in defining MDA with a broad scope of architectural concepts has the tendency of excluding them from conceptual trades. But the trades performed by IPT are limited in their ability to deal with uncertainty. In order to protect future developments, IPT members are implicitly encouraged to draw conservative conclusions which promote the status quo. Numerical analyses have the ability to perform a deeper uncertainty investigation due to their computational speed and automated exchange of information. This ability would enable more transparent investigations and could result in conclusions more open to new technologies. But as long as numerical analyses are excluded from architecture concept explorations their potential for deeper investigation will remain untapped.

2.3.4 **Deduction from Previous Descriptions**

In order to perform conceptual analysis, the disciplinarily-diverse architecting team must have the ability to integrate its heterogeneous perspectives. These perspectives are difficult to integrate due to the technical complexity and the level of depth necessary in the analysis. Two main approaches to the integration were described: Numerical model integration using MDA, and integrated product teams. Each approach has its strengths and weaknesses, but neither alone can offer an appropriate answer to conceptual activities. The fact that an architecture is a federated design problem due to its inherent technical complexity, implies that the experts must remain involved in the decision making process. Any methodological solution based solely on an integrated numerical model under the form of MDA will de facto exclude the experts from the decision process and will not provide viable solutions in a technically complex industrial context. On the other hand, methods based solely on integrated teams are limited in their scope of analysis due to their slow turnover and inherent conservativeness.

2.4 Linking Conceptual Design to Clustered Concurrent Preliminary and Detailed Design

Now that we have considered how the conceptual design activities are implemented, we shall focus on how they relate to subsequent design phases. In the previous section, we have observed that architecture definition is performed in the conceptual phase. The clustered concurrent engineering format is inappropriate to conceptual phases, but concurrent engineering approaches are. However, the clustered concurrent engineering (CCE) approach is essential to industrial developments in the preliminary/detailed phases.

We also observed that the definition of the architecture plays a central role in the definition of the design activities at the plateau-level. Therefore, to propose a valuable method for conceptual design, we must understand how conceptual design activities at the architecture level prepare, launch and steer CCE developments. In order to properly analyze this relationship, we shall consider the problem from a theoretical level and will observe how and why some of the traditional conceptual design approaches tend to fall short (or to be difficult to apply) in an industrial context.

2.4.1 **Theoretical Description of a Complex Design Process**

Previously in this chapter, design problems were defined under the form of an optimization problem. This problem presented in equation (1) is reproduced below:

$$\underset{X}{Max} \quad V(Att) \tag{1}$$

Subject to:

- $R \leq C(X, Spe)$
- $Att = f(X, Spe)$

Outcome:

Description of physical solution: X^* and Att^*

Based on this formulation and the observations made earlier in this chapter, we will now describe why the design of complex systems is challenging. We saw earlier that an aircraft is a systems architecture composed of a large number of elements. Based on this observation, we can say that the description of the aircraft requires a very large number of design variables. Using the mathematical formulation above, this particularity implies that the number of elements in the optimization variable *X* tends to infinity (i.e. $Base_X \rightarrow +\infty$). Previously, we also observed that the analysis of an aircraft is both uncertain and difficult to define. It is uncertain because it can be based on evolving technology or diverse expertise which can not necessarily be integrated easily (or practically). This aspect implies that the formulations of the equality constraints can not be defined explicitly (i.e. $V \approx f(Att)$, $Att \approx f(X, Spe)$, $C \approx f(X, Spe)$). Based on these observations we see that it is impossible to design a complex system solely from a holistic point of view. The number of optimization variables makes the problem an unfathomable task. Even if it was possible to identify all design variables, the formulation of the equality constraints would remain limited due to the complexity of setting up

accurate analyses. This theoretical explanation illustrates the impossibility to apply a pure concurrent engineering approach at all phases of the development.

2.4.2 **Architectural Design Role with Respect to Clustered Concurrent Developments**

These observations provide theoretical justification for clustered-concurrent developments. Clustered developments provide a breakdown of the complex system design problem into "simpler" and smaller subsystem problems. This breakdown is necessary to the completion of the development. It groups elements of *X* which relate to the same subsystem. These smaller groups of design variables are then determined by performing design processes at the subsystem level. These optimization problems are also easier to define with regard to their equality constraints.

2.4.2.1 Formalization of Subsystem Design

In the previous section we have demonstrated that the development of a complex system requires a decomposition of the overall design into subsystems to capture the scale and complexity of the design problem. The design problem at subsystem-level resulting from this decomposition is presented in equation (2). In this formulation, the "*i*" indices indicate that they qualify the subsystem "*i*". Note that the base of *Xi* is included in the base of *X*.

$$\underset{X_i}{Max} \quad V(Att_i, \Gamma_i) \qquad (2)$$

Subject to:

- $R_i \le C_i(X_i, Spe_i)$
- $Att_i = f(X_i, Spe_i)$

Outcome:

- Optimal subsystem description: X_i^* and Att_i^*

Also since we are dealing with a subsystem which is comparatively "smaller and simpler" than the system (i.e. the aircraft), the characterization of subsystem attributes

and capacity (*Atti* and *Ci*) is facilitated. But if this approach simplifies the expression of *Atti* and *Ci*, it must be based on a valid formulation of Γ*i*, *Ri* and *Spei*. As previously defined, *V* characterizes the value of the overall system. The system value is influenced by the attribute of the subsystem. For instance the value of an aircraft is influenced by the fuel burn of the engine. This relationship is characterized by the term Γ*i.* To illustrate the meaning of this term, these Γ*i* terms could correspond to the coefficient of a Taylor series of the function relating the subsystem attributes (e.g. engine fuel burn) and the overall value of the system (e.g. Net Present Value of the aircraft). The *Ri* and *Spei* are, respectively, the requirements that must fulfilled by the subsystem and the operating specifications at which the subsystem capability is delivered.

2.4.2.2 Objectives of the Architecture Design Problem

In order to formulate the subsystem design problem shown in equation (2), the architectural design process must formulate the general concept which will be pursued (i.e. identification of the optimal architecture). This formulation of the general concept will identify which subsystems are necessary. Based on this general concept, it is also necessary to define both, the context in which the subsystems are operating and the impact of subsystem attributes on the overall system value (i.e. the impact of the subsystem on aircraft performance/value). Based on this description, we see that the architecture design activity has three main objectives:

Objective 1: Optimize the architecture concept

Objective 2: Determine the operating conditions for subsystems

Objective 3: Relate subsystem attributes to overall system value

2.4.2.3 Formalization of the Architectural Design Problem

The first objective highlights that conceptual design is also an optimization process. In this optimization, the optimization variables are parameters describing the architecture (they will be noted as X_0). At this point, we are optimizing based on system-level metrics (V and R). From this perspective the conceptual design problem is similar to the fundamental design problem shown in equation (1). But the difference is now in the fact that since we could not optimize on the entire base of X, we are only determining the subsystem X_0.

In this formulation we recognize the constraints imposed by the fundamental problem: the capability of the aircraft must equal or exceed requirements; the value of the aircraft is a function the attributes of the aircraft. Since the optimization of some design variables (the X_i's) is now delegated to the subsystem-level developments, the definition of the aircraft attributes and capabilities must now include subsystem-level attributes ($Att_\bullet$). In order to determine what the subsystem attributes $Att_\bullet$ are, it is necessary to determine the operating conditions of the subsystems ($Spe_\bullet$ and $R_\bullet$) and the relationship between the value of the system and the attributes of the subsystems ($\Gamma_\bullet$). This necessity is what motivated the objectives 2 and 3 above. Based on these parameters, the subsystem design problem is formulated and attributes can be determined.

The formulation provided in equation (3) is theoretical but provides a formal description of what was described graphically in Figure 6 (reproduced below). In this figure, the "knowledge" flows provided by subsystem developments corresponded to the provision of $Att_\bullet$ to the program steering committee. The "guidance" flow represents the transmission of $Spe_\bullet$, $R_\bullet$ and $\Gamma_\bullet$ necessary to the formulation of the subsystem design problems.

$$\underset{X_0}{Max}\ V(Att)$$

Subject to:

- $R \le C(X_0, Att_\bullet, Spe)$
- $Att = f(X_0, Att_\bullet, Spe)$

(Subsystem design problem def.)

- $Spe_\bullet = f(X_0, Att_\bullet, Spe)$
- $R_\bullet = f(X_0, Att_\bullet, Spe)$
- $\Gamma_\bullet = f(X_0, Att_\bullet, Spe)$

(Resolution of subsystem design pb. – subsystem sizing)

- $Att_\bullet = f(\Gamma_\bullet, R_\bullet, Spe_\bullet)$

(3) Outcome:

- Optimal architecture description:

  X_0^* (Objective 1)

- Formulation of subsystem design pb $Spe_\bullet^*$, $R_\bullet^*$, and $\Gamma_\bullet^*$ (Objective 2 and 3)

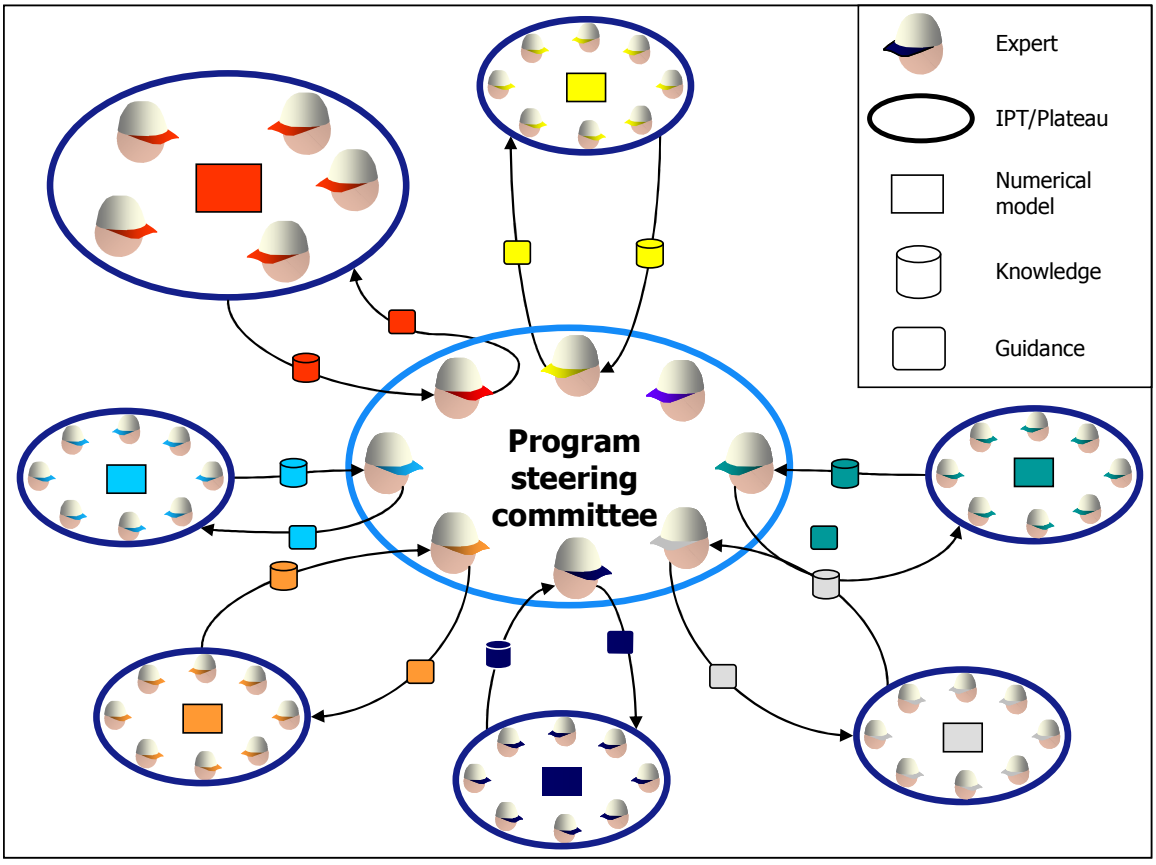


Figure 6 (reproduced): Clustered-concurrent development framework

2.4.3 **Consequences of Conceptual Design Approaches on the Detailed Design Problem Formulation**

As the architecture is explored, it is neither practical nor possible to perform full blown subsystem developments for each exploration. In this context, the theoretical formulation of the architecture design problem can not be used. The following section will describe some common paradigms in conceptual design and will describe how they relate to subsystem development.

2.4.3.1 Description of the Traditional Conceptual Design Approach

In order to simplify the conceptual design process, a common approach is to consider that the subsystem will be a solution which can be scaled up and down with respect to some scaling attribute ($Att'_{\bullet}$). In this context, the notion of goodness of the subsystem solution is lost which implies that the conceptual design problem is reduced to equation (4). The notion of value at the subsystem level has disappeared from the subsystem analysis.

$$\underset{X_0, Att^*_{\bullet}}{Max} \; V(Att)$$

- Subject to:
- $R \le C(X_0, Att_{\bullet}, Spe)$
- $Att = f(X_0, Att_{\bullet}, Spe)$

(Subsystem design problem def.)

- $Spe_{\bullet} = f(X_0, Att_{\bullet}, Spe)$
- $R_{\bullet} = f(X_0, Att_{\bullet}, Spe)$

(Subsystem design approximation)

- $Att_{\bullet} = f(R_{\bullet}, Att'_{\bullet}, Spe_{\bullet})$

(4) Outcome:

- Optimal architecture description: X_0^*
- Formulation of subsystem design pb.: $Spe^*_{\bullet}$, $R^*_{\bullet}$ and $Att^*_{\bullet}$

To illustrate this approach, let us consider the sizing and synthesis approach for aircraft conceptual design proposed by Mattingly [25]. In his approach the architectural design variables are parameters like the type of engines (turbojet/turbofan/turboprop), or type of wings. In this design approach the wing and engines are designed and scaled based on two scaling attributes ($Att'_{\bullet}$). For the wing the scaling attribute is the surface area (or wing loading) and for the engine the sea level static thrust for the engine. Based on these scaling attributes, the other attributes of the subsystem are adjusted to operating conditions to determine the capacity of the aircraft to perform the requirement. For instance, the wing lift and drag (which would be considered respectively as a capability C and attribute Att of aircraft) are determined from the polar curve and surface area. Similarly the engine size and flight conditions define the thrust capacity of the aircraft. This subsystem attribute is used to determine the thrust required from the engine (R). The constraint plot resulting from this evaluation (using the energy equation) is in fact the visualization of the constraints imposed by $R \leq C$. The sizes are determined by determining the smallest engine and wing able to perform the requirements. This implies that together the size attributes of the wing and engines define the value of the aircraft (i.e. parameter V).

2.4.3.2 Difficulties Implied by Traditional Approaches in a Subsystem Design Context

The approach highlighted above provides a simple and direct means to perform the optimization at the aircraft-level. But we can see that it does not support objective 3. Indeed it does not relate the subsystem attributes to the value of the integrated system (i.e. aircraft value). As a result the design problem is reduced to a design to target problem where the targets are defined by the estimated subsystem attributes used in conceptual design.

$$\underset{X_i}{Min} \quad \sum_{Att^*_i} \max\left(0, Att_i\left(X_i, Spe_i\right) - Att_i^*\right) \tag{5}$$

Subject to:

- $R \le C\left(X_i, Spe_i\right)$
- $Att = f\left(X_i, Spe_i\right)$

Outcome:

Optimal subsystem description: X_i^* and Att_i^*

To illustrate this situation we can considered, the engine design approximation in Mattingly's sizing and synthesis approach. The weight (which belongs to the $Att_\bullet$ category) will be derived from the engine scaling parameter ($Att'_\bullet$). Similarly the efficiency and therefore engine fuel burn will be defined based on similar models. These estimated attributes (engine weight, efficiency, etc…) are important factors in the overall performance of the architecture. Therefore, in order to ensure that the engine design is going to allow the aircraft to meet performance requirements, it is necessary to make sure that the attributes of the engine match those predicted in conceptual design. In other words, the attribute estimates in conceptual design become the targets for subsystem developments.

2.4.3.3 Subsystem Intrusive Conceptual Design Approaches

Previously we have observed that in conceptual design we reduce the base of the design variables pertaining to the fundamental problem (*X* in equation (1)) to a subset limited to architecture-level design variables (*X0*) in equation (3)). The remaining design variables are delegated to the subsystem developments. Some research efforts [26] propose to increase the size of the base of *X0* by integrating subsystem-level design variables in conceptual design. If we compare Dr. Drela's approach to what is traditionally done with Mattingly's approach, we will observe that these methods differ primarily with regards to the type of design variables used in the optimization. If Mattingly limits the tradeoffs to a single attribute for each wing and engine, Drela will include several parameters for both the engine (e.g. pressure ratios, temperature , bypass, etc…) and the airframe (e.g. wing and fuselage geometry parameters). Using this approach Drela is able to explore subsystem trade-offs at conceptual design phase. This method is limited in three important ways.

The first important limitation is the complexity in setting up the analysis associated with the proposed method. In order to provide meaningful results, it requires subsystem models which capture the performance of the finalized systems. If they are not, the optimum design found in conceptual design will be suboptimal (or worse, infeasible) once developed in detailed design phases. In other words, this method allows for more accuracy by considering more subsystem design variables. This accuracy is gained at the expense of the practicality in setting up the model.

The second limitation concerns design freedom left for subsystem design studies. In Figure 3, we saw that it is preferable to maintain the design freedom until more knowledge about the design problem has been established. Conceptual design is located early in the design process when knowledge about the technology is evidently low. Therefore, even if the model used includes multiple disciplines and includes some depth into the definition of the subsystems the degree of knowledge and visibility on subsystem

details is necessarily limited. Therefore increasing the base of the design variable set used in conceptual design allows performing more trades at the conceptual design level, but at the same time limits the design freedom at the subsystem level.

The third important limitation is the fact that, by including subsystem design parameters, the study will prescribe design decisions which do not pertain to the scope of the designer control. In other words, if the study is performed by airframers, the designer will not have direct control over the design parameters of the engine. The airframer has neither the industrial responsibility nor technical visibility to determine the optimal pressure ratios, and temperatures of an engine. The engine maker has this responsibility since he is the one accountable for the development and production of the engine. As a result, the implementation of this method is limited in a competitive and intellectual-property-protected industrial context.

2.4.4 **Problem Associated with Traditional Conceptual Methods in an Architectural Context**

2.4.4.1 Distinction between Targets and Objectives

Designing to targets is convenient as it allows for the formulation of discrete and independent criteria for the attributes of subsystems. Using targets is very practical for large problems for several reasons. First of all, a target is discrete and unlike an objective it is either met or failed. Secondly, when targets are used in a multi-attribute problem they isolate the performance of the subsystem on each attribute. For these reasons, driving subsystem developments based on design targets rather than the full optimization problem (including the objective function) is certainly more practical.

However, using targets implies several important assumptions. In order to distinguish the nuance between an objective and a target we should consider their meaning under the form of utility. The term utility, in the context of this discussion,

corresponds to the degree of "goodness" of the alternative. Utility takes values comprised between zero and one, where zero corresponds to an absolutely unacceptable design and one to the perfectly ideal design. Let us assume that a design attribute has an importance with respect to the requirements (example: operational range). If the requirement is expressed as a target (or non-negotiable requirement) the relationship between the attribute and the utility behavior will be discontinuous (see Figure 15).

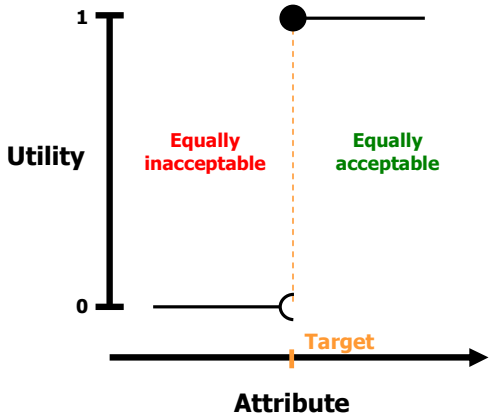


Figure 15: Utility implied by a target

The utility follows a Boolean behavior, where any points below target are equally inacceptable and all points above are equally acceptable. There are no nuances beyond these two levels and the degree by which the target is exceeded or failed is not considered. Let us assume the attribute represented above is the operational range of an aircraft with a target of 6000 nm. If we consider two aircraft alternatives, one at 3000 nm and another one at 5500 nm, both will be considered as equally unacceptable. In a similar fashion, if we consider two aircraft, with 6000 nm and 8000 nm respectively, they will be both considered as equivalent. In other words, using a target to capture requirements provides a stiff formulation of the problem. Formulating the requirement as an objective introduces nuances in the perception of value.

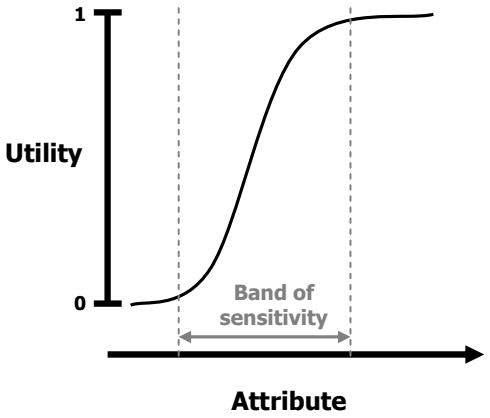


Figure 16: Utility associated to an objective

Figure 16 represent an example of a relationship between the attribute and the utility. The shape of the relationship presented is arbitrary, but it is important to notice that its behavior is continuous. Therefore an objective implies a level of sophistication which allows capturing the nuances in the goodness of the various alternatives considered. In the example for the range requirement, the aircraft with a range of 5800nm will no longer be dismissed but would still be penalized compared to an aircraft capable of flying 6000 nm. We can also see that the objective has also a band of sensitivity which makes it equivalent to a target on extreme values of the attributes. In the example for the range, the band of sensitivity would start at the range value where no airline would buy the aircraft because the range is too small. On the other side, the band ends at the range value where the additional range would no longer matter to the operator. In Figure 16, we can see that outside the band of sensitivity the utility is held at the value zero on the left side and unity on the right side.

The use of a target eliminates the complexity associated with defining the relationship between the attribute and the notion of value. But this simplification is done at the expense of the nuance described earlier. There can be no general rules in terms of what should be considered as a target versus an objective, but when a choice is made, it is important to understand what is being given up. In general it is preferable to limit the use

of targets to requirements implying a narrow band of sensitivity. For example, requirements related to certification are arguably the most non-negotiable requirements because you can only pass or fail certification (no nuances there). To some degree, requirements pertaining to system of systems integration (ex: airport integration – landing field length) may be considered as target as the aircraft or its interfaces are either compatible or incompatible (it can land on the runway or it cannot).

2.4.4.2 Illustration of Improper Formulation of Subsystem Design Problems

The conceptual design activities occur at the beginning of the development. For a commercial aircraft programme, conceptual trades may occur 10 years before entry into service. Therefore some degree of uncertainty is unavoidable when evaluating alternatives in a conceptual context. This uncertainty will lead to the definition of targets that may turn out to be unreachable or below the capability that can actually be achieved. Formulating a design problem using targets can only qualify whether a design is acceptable or not. This binary value framework does not provide guidance for the substantial amount of trade-offs that must be performed at the subsystem level. To illustrate this point one can consider the engine design. Although the engine is a subsystem to the aircraft, it is not a non-complex (i.e. simple) design problem. Many design variables will strongly influence the value of the engine (by-pass ratio, compressor and fan ratios, etc…). Setting targets defines a space of acceptable solution. But it does not guide the designer within this space since it will not distinguish the best of multiple acceptable solutions. Also, if no solutions are acceptable based on the targets, the designer is left with no guidance toward a solution which will limit the impact of the underachievement of his/her design.

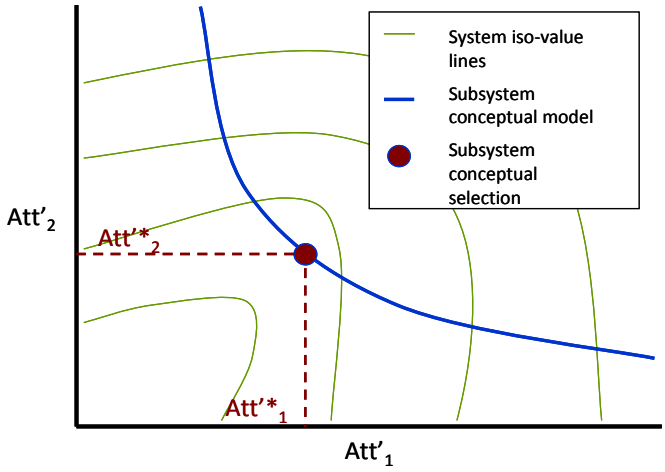


Figure 17: Notional representation of conceptual design

In order to illustrate the processes described previously we shall represent the optimization problems as Pareto fronts. All designs located on the front are compliant with the constraints listed in equation (4). The coordinates of the graph represent notional attributes. These two attributes correspond to the attributes of a subsystem (this representation assumes that attributes belonging to other subsystems were already defined and fixed). These attributes have an influence on the overall system value. Their influence is depicted by the system iso-value line (in green).

If the conceptual design problem is optimized properly, the outcome of the design will be the selection of attributes that will maximize system value (represented by the red dot). If a traditional conceptual design process is carried on, it will use this optimal value to allocate *Att'*1* and Att'*$_2$ as targets to the subsystem developments. It is important to note that these values are at this point predictions based on the conceptual model for the subsystem (gross approximation); but will become hard constraints which will be used to guide subsystem developments.

On the other hand, no information qualifying the topology of the system value with respect to *Att'1* and *Att'2* is preserved. Consequently, as detailed developments are performed, the subsystem designers will have no further information beside targets

imposed previously by Att'^*_1 and Att'^*_2. Their design problem can therefore be presented by the following figures.

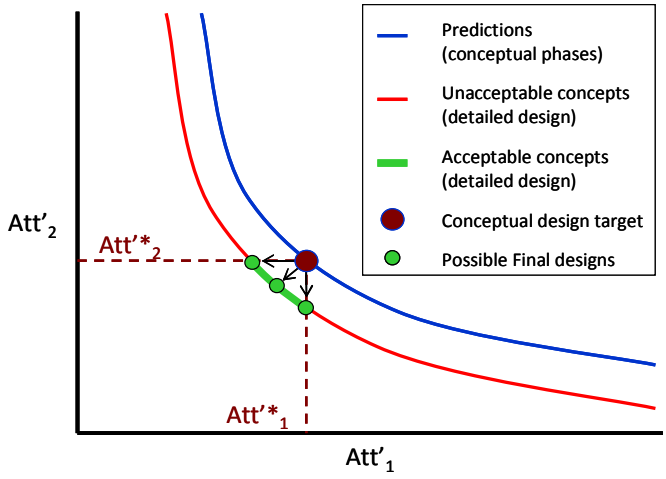


Figure 18: Notional subsystem design after conservative conceptual design predictions

Figure 18 presents a situation where subsystem performance was underestimated in the conceptual phases. Therefore, developments of subsystems will provide a set of solutions more favorable than expected. These "more favorable" solutions are presented as a new Pareto front (shown in green and red). The subsystem designs depicted by this new front satisfy all constraints listed in equation (5), but only the green section will satisfy the targets. Hence, we can see that if the subsystem designer was to blindly proceed toward the conceptual target, his/her design would be suboptimal. But the problem definition does not provide any specification in terms of the best solution amongst the green solutions.

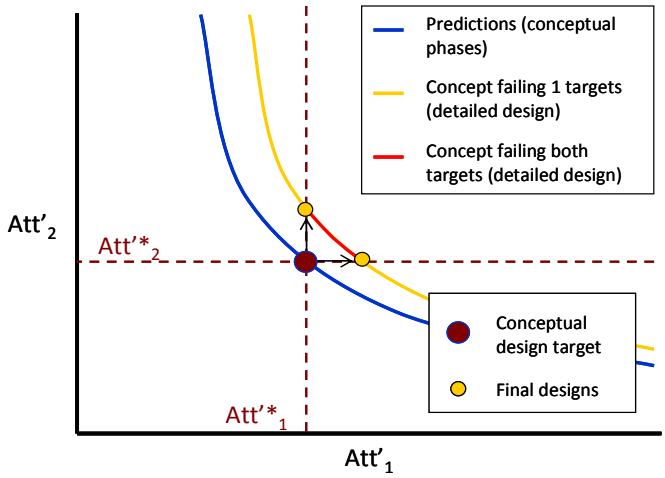


Figure 19: Notional subsystem design after optimistic conceptual design predictions

Figure 19 shows a situation where the conceptual predictions were too optimistic. In this case, no design can meet all targets. Given the problem formulation provided by equation (5), the designer has no information on how to palliate the underperformance of his/her design. Therefore, the designer is likely to relax one of the targets in order to meet the others (as shown by the orange dots in Figure 19).

2.4.5 **Deduction from Previous Observations**

Conceptual design sees the genesis of the Aircraft System Architecture (ASA). This genesis defines the main lines development that will be pursued in later design phases. To support this process of creation, an architect and the experts (forming the architecting team) propose solution concepts, analyze them and compare them. The conceptual design phase's objectives are to:

- Select the "most appropriate" architectural concept
- Define a framework for preliminary and detailed design phases.

The design framework must provide guidance for the clustered concurrent design approach. With this framework, the sub-design problems corresponding to each subsystem should be as loosely coupled as possible and the local optimization should

contribute to the optimization of the architecture as a whole. In order to facilitate preliminary and detailed design phases the framework needs to:

- Define an optimal architecture concept.
- Determine the operating conditions for subsystems (requirements and operating specifications)
- Relate subsystem attributes to overall system value.

2.5 Research Objectives and Research Questions

In this chapter, we have developed the fact that concurrent engineering is necessary to streamline the industrial development process of an aircraft. In its implementation of concurrent engineering precepts, the industry was able to deploy a development framework which supports concurrent developments of detail design phases. This framework, referred to as clustered concurrent engineering (CCE), is based on multiple design groups, each dedicated to the development of a specific subsystem design perimeter.

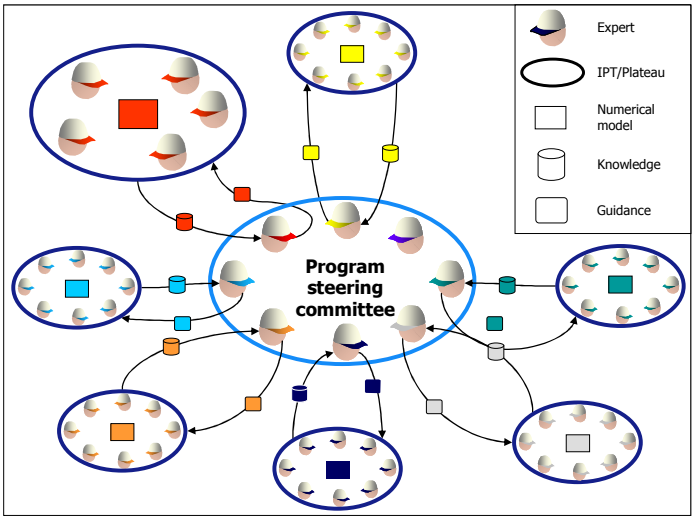


Figure 6 (reproduced): Clustered-concurrent development framework

But the initiation of this process requires the definition of clear design perimeters and guidance for their developments. When few architecture changes are required since the previous aircraft generation, the CCE framework can be defined by previous

development experiences and adjustments from a program steering committee. As the aircraft system architecture is fundamentally changed, the adjustments necessary to the CCE framework require technically challenging conceptual trades at aircraft level beyond the current role and capacity of a steering committee. To perform these trades, new collaborative architecting methods are necessary. From this observation we can formulate the following objectives:

Objective 1: Provide a flexible architecting method to allow for the absorption of technological opportunities.

Objective 2: Definition of a strategic plan for the industrial development of the architecture.

In this chapter we have observed that two approaches could be used to perform technical trades with multi-disciplinary knowledge:

- Integrated product teams (IPT)
- Multi-Disciplinary Analysis (MDA).

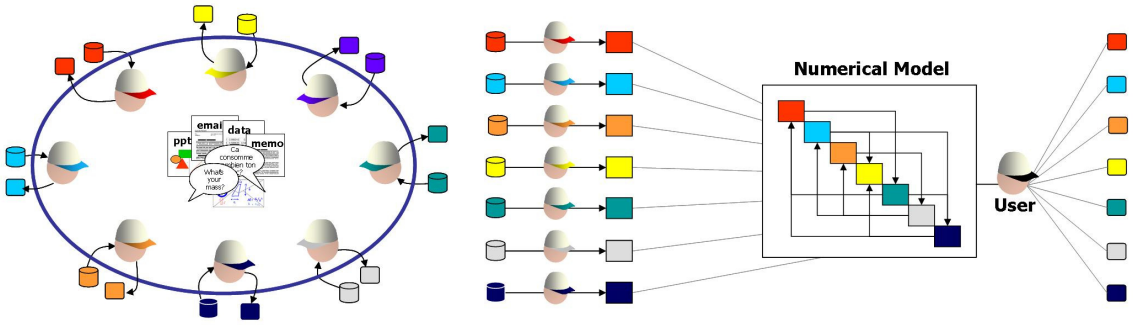


Figure 20: Comparison of IPT and MDA

The IPT is based on the collaborative work of a group of experts. This approach is slow, sometimes arbitrary in its conclusion, but adaptable. The MDA approach is based on the integration of numerical models. This approach is complex to setup but fast to execute. These observations lead us to the following Research Question:

Research Question 0: How can we structure a conceptual design framework supporting the following objectives?

Objective 3: Accelerate turn around in architecture concept analysis

Objective 4: Detect architectural opportunities.

Objective 5: Prepare a framework guiding detailed design developments

We have observed that neither the IPT nor MDA approaches can achieve these goals alone. But each has the ability to contribute to the solutions. Therefore the fundamental assertion on which this thesis is based is:

The objectives can be achieved using an IPT to conduct conceptual design trades, based on an analysis performed by an MDA composed of numerical models defined and controlled by the experts composing the team.

A preview of the framework suggested by the hypothesis is presented in the following figure.

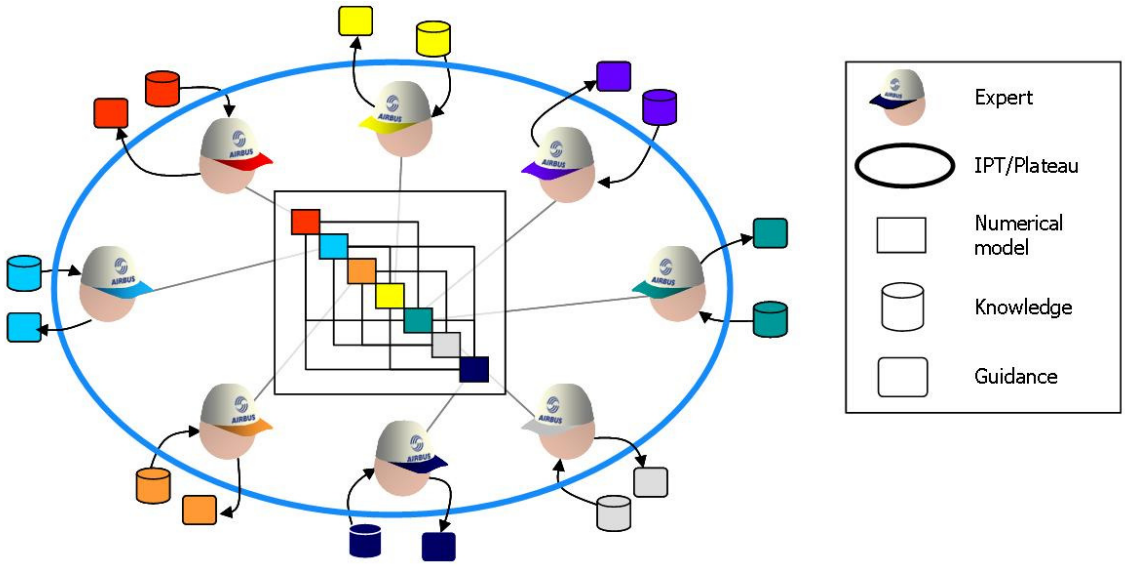


Figure 21: IPT MDA based approach

In conceptual design, the starting point of all investigations is the mission. For a commercial aircraft, the needs are very diverse and complex to define. We saw previously that this complexity in needs and complexity in solutions is the primary cause and motivation for the use of architecting techniques. Therefore the first research question for this thesis will be:

Research Question1: How can we characterize the mission in a fashion that captures all requirements driving subsystems sizing?

Based on the mission definition, architectural solutions must be defined. As we saw earlier, the IPT has the expertise necessary to define solutions but the MDA offer the means to enable the analysis process. The amount of time and efforts necessary to setup an MDA remains a barrier against its deployment in conceptual architecture design. Therefore, in order to reach objective 3, the following gap must be addressed:

Research Question 2: How do we facilitate the set-up of MDAs representing a broad variety of architectures?

Conceptual sizing practices have a tendency to neglect the potential for subsystem coordinated optimization leading to architecture performance improvement. Neglecting this level of trade results in conservative conclusions as the traditional and well optimized architecture is compared to a new architecture composed of inappropriate subsystems.

Research Question 3: How can we mathematically formalize subsystem sizing while allowing for their coordinated optimization?

One of the most important observations of this chapter is that a good architectural development is not only a good architectural concept. It is before all, a development where subsystem design activities contribute to the optimization of the architecture as a whole.

Research Question 4: How do we translate architecture level objectives and lessons learned during architectural trades into specifications and guidelines to the subsystem developments?

Chapter 3

State of the Art Review and Observations

The previous chapters have set up the industrial context of this thesis and formulated the problem targeted by this work. This chapter will now focus on the methodological and technical aspects of the problem. This chapter's objective is to review and explain the solutions relevant to the problem proposed by the research community in the field of conceptual design, systems engineering and architecture design. This objective includes understanding how these methods can or should be applied and what their shortcomings are when applied to ASA conceptual design. To support this objective, the description of most methods includes an overview of the assumptions on which they were constructed. The analysis of their assumptions will help us understand their limitations and pinpoint the technical gaps that must be addressed by the thesis. This review will be organized around the different activities constituting the process of conceptual design of an ASA. The process is decomposed into three main activities: requirement definition, architecture definition, and concept analysis. The three activities are represented in the flow chart in Figure 22.

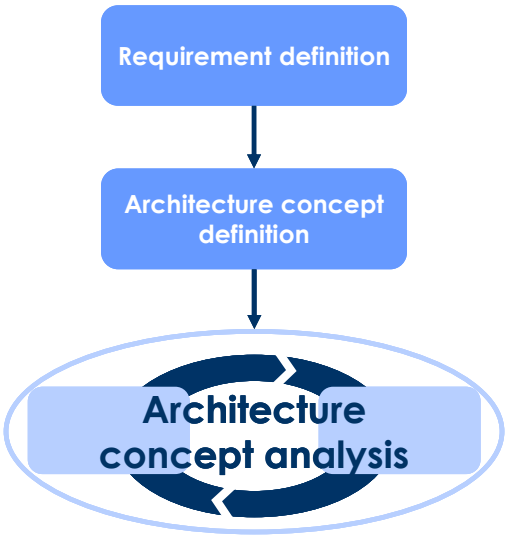


Figure 22: Activities constituting ASA conceptual design

The requirement definition activity corresponds to the classification of the needs driving the design of the architecture. These requirements motivate the definition of an architecture concept. This definition selects the components composing the architecture and the structure in which they are integrated. This definition specifies the concept that must be analyzed. The state of the art review is organized around these three phases. The first three sections of this chapter will each focus on a different phase of the conceptual design process. This chapter will be concluded by a fourth section comparing existing methodologies which cover the three phases of design.

3.1 Methods for Requirement Formulation

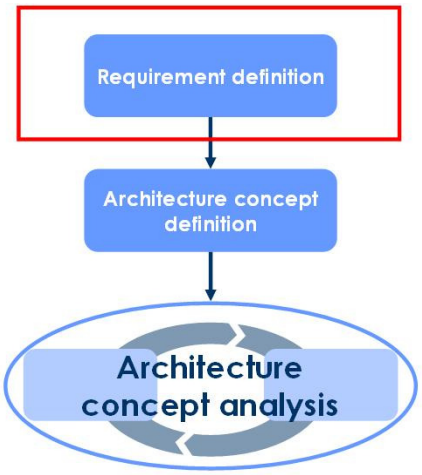


Figure 23: Requirement definition phase

This thesis does not attempt to formulate solutions to analyze the needs of the customer. Therefore the discussion in this dissertation will assume that the needs are predefined under the form of a mission, certification requirements, and a set of services implied by the aircraft mission as a passenger transport vehicle.

In the phase of requirement definition we are setting up the design problem for the architecture conceptual developments. The objective is to organize the general requirements and needs in a fashion that will clearly and unambiguously motivate the development of the architecture. The design problem is constituted of requirements which

qualify the non-negotiable needs from the stakeholders, and a value framework for evaluating the "goodness" of the concept alternatives. It is important that the requirement formulation phase capture both these aspects of the problem.

The design problem is what drives the formulation of a conceptual solution. If the formulation must facilitate the definition of the architecture concept alternatives, it should be in a form that is indicative of solutions without constraining its scope.

In the background research performed in the field of requirement formulation, several methods were identified as being instrumental to the formulation of the design problem for architecture conceptual design. The techniques and tools that will be presented herein were grouped in two families of solutions:

- Functional analysis
- Objectives analysis

3.1.1 **Functional Analysis**

Interesting methods were proposed with functional analysis methods. The field of functional analysis corresponds to the ensemble of methods and techniques involving the study of functions. This field originates from industrial engineering studies on Value Engineering [27].

In a design context, the concept of function vehicles two notions:

- The notion of purpose for an object
- The notion of an action or a role within a context for this object

A function is generally described by a phrase which includes a verb sometimes qualified by a noun. An example of a function can be: "Propel" or "Provide thrust". Functions play a critical role in formulating the need for the architecture as it brakes down the bulky aircraft mission. The breakdown makes a bridge between the mission formulation and the physical formulation which is specified as part of the architecture concept definition.

3.1.1.1 Functional Tree

The mission breakdown can be represented under the form of a **functional tree**. The functional tree is the most widespread application produced by the functional analysis field. The meanings for the branches and leaves of the tree are represented Figure 24 [27].

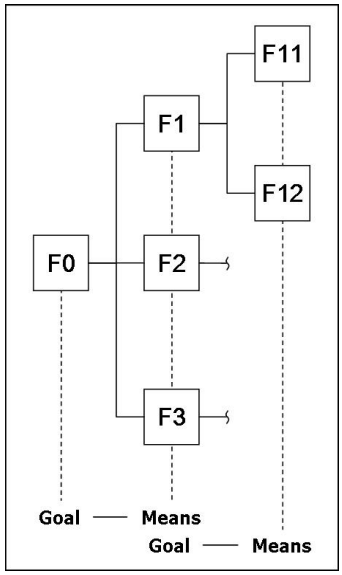


Figure 24: Function Family Tree principle

The function at the root of a group of leaves represents the main function which motivates the breakdown. The leaves coming out of this function represent the functional means by which the function is achieved. They correspond to the sub-functions both implied and necessary to the achievement of the main function. The breakdown process of functions is essential as it simplifies and often clarifies general functionalities which are not indicative of a specific physical solution. The mind-map represented in Figure 25, presents an example of a functional breakdown for a commercial aircraft.

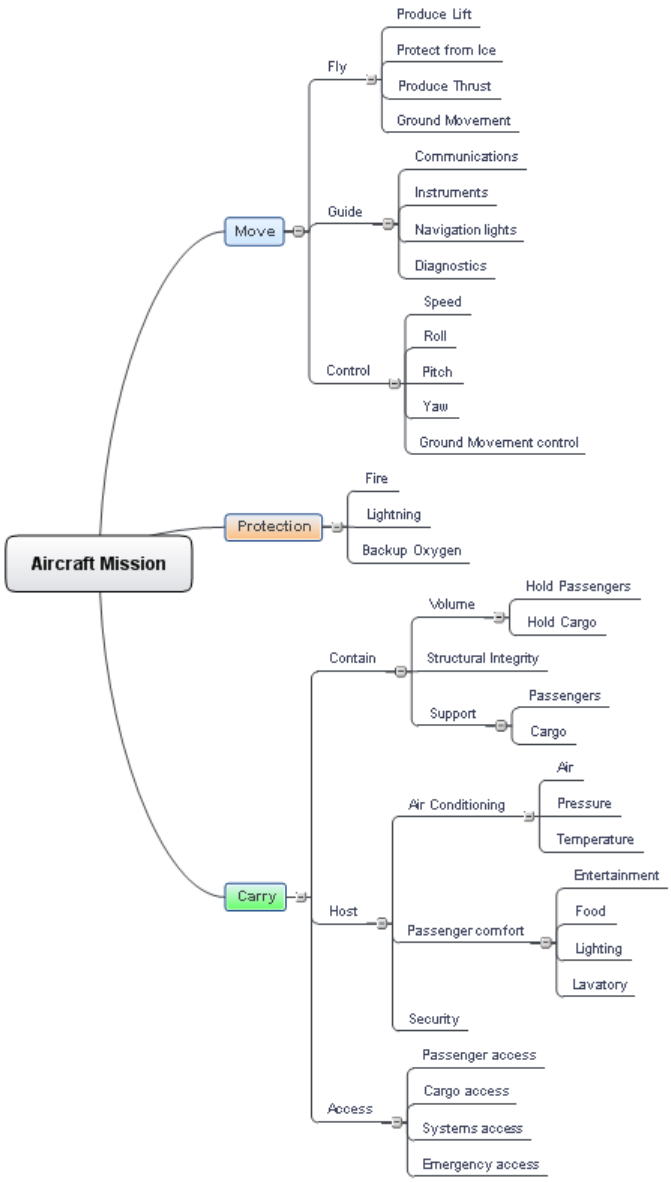


Figure 25: Notional functional breakdown of aircraft functions[28]

3.1.1.2 Classification of Functions

If we look closely at the tree in Figure 25, we can notice that critical functionalities were omitted. For example, there are no functions specifying that electricity should be provided. Generally, electricity is necessary to most of the functions listed in the tree. But at the same time, if we strictly consider the mission of an aircraft,

nothing directly requires that electricity should be provided. These functions are therefore very difficult to allocate to a specific branch of a tree.

From this observation we can observe that multiple types of functions are present within the architecture. The functions listed in Figure 25 are referred to as **boundary functions** by Mavris et al. [29] or basic functions by Akiyama [27]. The term boundary refers to the fact that they provide a link between the mission and the physical architecture. This type of functions is at the boundary between the mission and the physical boundary of the architecture. On the other hand, functions like "provide electric power" correspond to another type. Since these functions are not directly required by the mission, Akiyama refers to them as "secondary functions". As we will see in more detail in the next section, secondary functions are always induced by the physical solution retained to implement a basic function. Therefore, Mavris et al refer this later type of functions as "**induced functions**". For practical purposes in this dissertation, the terms boundary function and induced function will be used.

3.1.1.3 Function Analysis System Technique (FAST)

The Functional Analysis System Technique (FAST) is a functional analysis method introduced by Charles W. Bytheway [30] which includes both boundary and induced functions. The FAST diagram defines the relationships between the functions, using the convention presented in Figure 26.

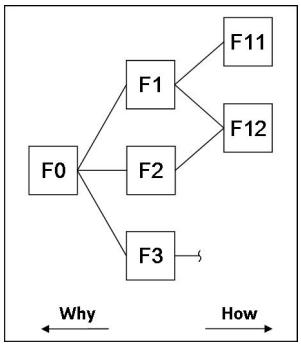


Figure 26: FAST diagram principles

The principles of FAST are based on the observation that as a function is implemented by a physical solution, a new supporting function is necessary. This supporting function may be shared by multiple physical solutions. Therefore if we consider the FAST diagram, the functions at the left (which correspond to the boundary functions) justify "Why" the functions to the right (induced functions) are necessary; and vice versa, the function at the right explain "How" the functions to the left are implemented. Since the diagram is not a tree, consequential functions (or induced functions) can now be shared amongst originating functions.

The nature of the FAST diagram allows a complete functional description of an architecture. But due to the fact that it includes induced functions which are motivated by physical elements, the FAST diagram does not provide a universal representation of the mission. For this reason, when FAST diagrams are used, it is important to acknowledge that some aspects of this representation are architecture specific and do not necessarily provide a universal functional breakdown of the mission.

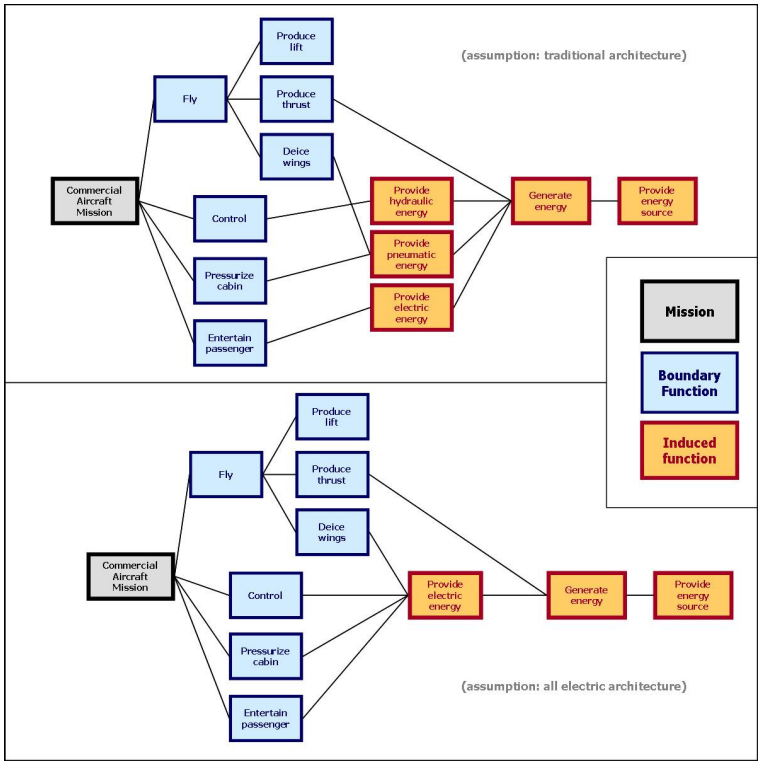


Figure 27: Example FAST diagram

To illustrate the point above one can consider the FAST diagrams represented in Figure 27. They provide a partial description of two commercial aircraft system architectures. One is for the all electric aircraft, the other for the traditional aircraft.

3.1.1.4 Quantitative Use of Functions

The needs imposed by the mission requirements provide the characterization of its functions. The function is descriptive of a role. In order to relay the requirements implied by the need, functions must be quantified. This quantification is simply referred to as a **functional requirement**. If we consider the function "Produce thrust", the functional requirement could be "10,000 pounds".

In order to complete the description of the function the concept of operational context under which the function must be achieved is necessary. The **operational specification** in the thrust example above would be "at take-off at sea level in ISA conditions. The association of the function, its functional requirements and its operational conditions constitute what is defined as **functional specifications** [1]. Therefore, an example of a functional specification would be: "At take-off regime at sea-level in ISA conditions, produce 10,000 lb of thrust". This formalism provides a mean to organize and keep track of target requirements (example the engine shall provide 10,000 pound of thrust). Therefore, functions allow representation of "non-negotiable" need.

3.1.1.5 Synthesis of Functional Analysis Techniques

Functional analysis provides means to decompose the mission into more explicit and technically-manageable pieces of information. The functional tree facilitates the classification and breakdown of functions. It allows bringing down the decomposition of the mission requirements to a level of granularity which can guide architecture concept definition.

The functional analysis of the mission alone can not provide simultaneously a complete and universal description of the functions necessary to the fulfillment of the

mission. The functional tree is limited to boundary functions which are universal, but provide an incomplete description of the functions necessary within the architecture. On the other hand, FAST diagrams have the potential to represent all functions present in the architecture, but their description is not universal as it implies the physical implementation of some functions.

The quantitative aspects of requirement can be relayed by the functional perspective through the formulation of functional specifications. Functional specifications describes: the context in which the function is performed and provide a quantitative and qualitative description of the functional requirements. The functional formulation, however, does not apply well to the negotiable aspects of some requirements.

3.1.2 **Objective Analysis Methods**

In the previous chapter we have described the tendency in the industry to drive architectural conceptual design by performance and cost targets (attribute allocation). Using this approach implicitly defines assumptions on what the goodness of the solutions should be and influence the outcome of the study. Performance and cost are tradable attributes (i.e. more cost may be acceptable for more performance and vice versa). Placing targets assumes that any concepts with a lower cost and higher performance than target are equally acceptable. This assumption may be valid under some very specific marketing assumptions: we have clients with inflexible budget and insensitive to over-performance. In a detail design phase were purchase agreements have been signed already, these assumptions may be valid: The customer is committed to a price and the performance of the delivered aircraft will no longer influence the purchase decision. But in early design phases where the baseline concept is still open and the airframer is trying to define the best product which will attract orders, anything related to cost or performance is negotiable and therefore part of an objective function. Therefore one can say that the use of tradable objectives allows making the link between the marketing

motivation of the program and the engineering development aspects. The use of targets simplifies the design decision making process but does not enable this link. Based on this description, we shall now discuss the possible methods associated with the formulation of objectives

3.1.2.1 Definition of Objectives

In a design context, an **objective** is "a direction in which we should strive to do better" [10].It is formulated as a verb, an object, and possibly a qualifying phrase [31] for example: "Make the aircraft safe" or "Minimize cost".

An objective will always refer to an attribute of the design. This **attribute** is a parameter specific to a concept or an alternative which contributes in describing its nature. When an attribute is the object of a design objective, we will refer to this attribute as a **Measure of Effectiveness** or **Figure of Merit** (MoE or FoM). In the design objective "minimize cost", the FoM is cost.

When an objective is formulated, there is often a notion of directionality involved. Directionality indicates whether smaller or larger values of FoM are preferred. The FoM and directionality will be used to guide decisions in the design process. They are not always specified explicitly by the formulation of the objective. In order to facilitate the design process in later phases, it is preferable to formulate the objective as "Minimize/Maximize FoM". For instance, the objective "make the aircraft safe" should be formulated as "maximize safety".

3.1.2.2 Objective Tree

In order to represent the objectives identified as important in the study, a technique similar to the functional tree presented earlier has been proposed [32] , [31]. This technique allows decomposing and relating the previously identified objectives within a tree. The principles used in building the tree are represented in Figure 28.

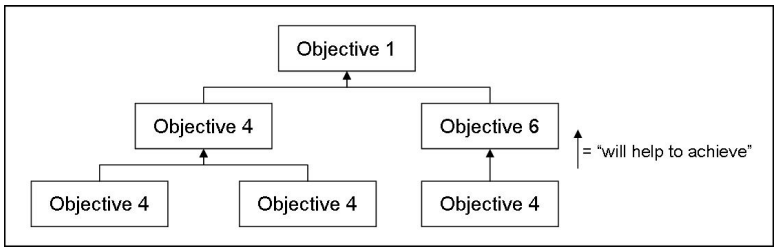


Figure 28: Objective tree

A notional objective tree is represented in Figure 29. This technique is particularly useful in determining if an objective is a "fundamental" objective or if it is a "means" objective. A **means objective** is an objective which is pursued in support of another. The **fundamental objective** is the one that can not be explained by another objective [10]. For example, the objective to minimize the empty weight of the aircraft is a means objective. It is a means objective because the weight of the aircraft per se is not what really matters. But since we know that, the lighter the aircraft is, the less fuel it consumes, then we can say that weight minimization is a means objective to minimize fuel consumption. On the other hand, if we consider the objective at the left hand side of Figure 29, Maximize program Net Present Value (NPV), this objective can be considered as the most fundamental objective because it includes all the other objectives and can not be explained by any of them individually.

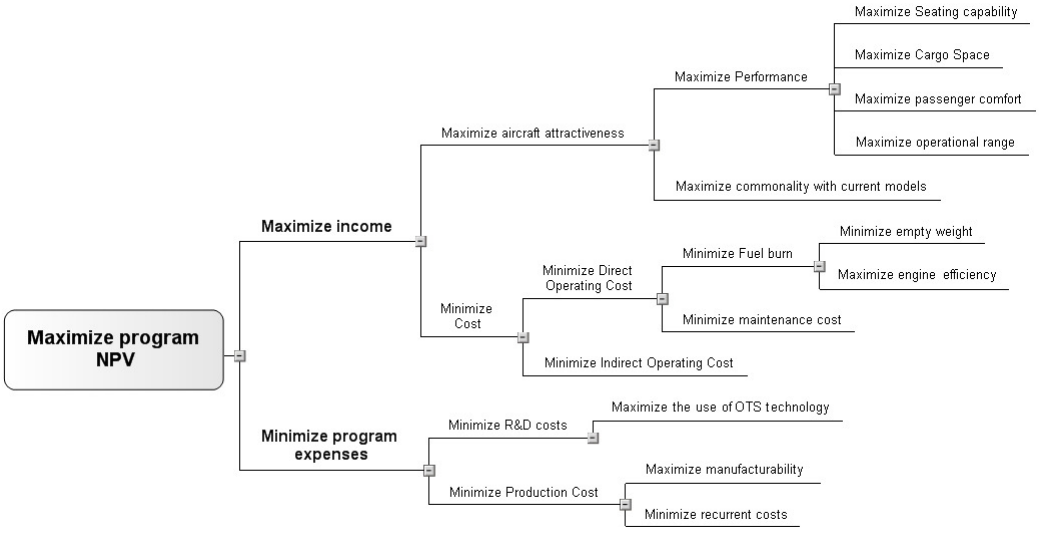


Figure 29: Notional objective tree for an aircraft development program

We can also notice that the FoM corresponding to the objective at a node (e.g. fuel burn), is always a function of the FoMs of its leaves (empty weight and engine efficiency). Therefore the consideration of the most fundamental objective overrules the consideration of its means objectives which are often conflicting with one another. The means objective FoMs are often more easily traceable to the trades than fundamental objectives FoMs. Therefore, it is generally more practical to consider means rather than fundamental objectives. The example of aircraft weight optimization is a good illustration. As a subsystem designer is developing his/her system, he/she believes that optimizing the weight of his/her solution will contribute to the fundamental objective. But means objectives can be misleading. For example, optimization of weight at the expense of other objectives (e.g. energy efficiency) may result in suboptimal solutions from a fundamental objective perspective. Some of the conclusion of the POA research program lead by Lester Faleiro [23] raised concerns about the fact that designers in the industry tend to focus too much on weight at the expense of more fundamental objectives.

3.1.3 **Overview of Requirement Formulation Methods**

It is difficult to define the aircraft design problem in a way which will guide the development of the system architecture without constraining it. The literature review in this field has shown that methods based on objectives and functions analysis have been proposed. These techniques allow an organization of the requirements into functional specifications and objective formulations. This organization breaks down the overall design problem into smaller problem. In the design of architectures where different experts will focus on different aspect of the problem, having this breakdown allows both the identification of subsystem level solutions (using functions) and the comparison of alternatives at aircraft level (using objectives).

The literature review has also revealed the fact that functions can be classified under several categories (boundary functions and induced functions). The universality of

the induced functions is conditioned by choices made in the physical implementation. Therefore it is impossible to have simultaneously a complete and universal functional breakdown.

This dissertation focuses primarily on power architectures. The power architecture provides for the functions related to energy generation, transformation and distribution. These functions are all classified as induced functions given the fact that they are induced by physical implementations of aircraft functions (or boundary functions). As a result, functional analysis techniques are limited in their potential to help in organizing the requirements of power architectures. Either they will provide solutions that are complete, but conceptually restrictive (FAST technique) or imprecise and incomplete (functional trees).

Previously, we have also observed that the requirements can be formulated in two ways: functional requirements and objectives. Functional requirements tend to correspond to non-negotiable requirements and objectives to requirements which enable the comparison of alternatives. We will now proceed to the review of methods and techniques dedicated to the definition of architecture alternatives.

3.2 Methods for Architecture Concept Definition

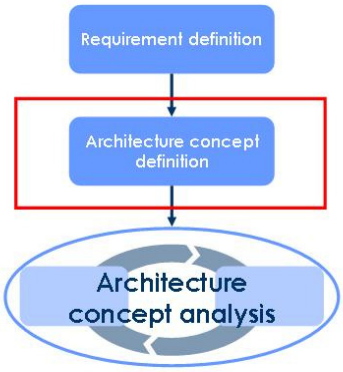


Figure 30: Architecture concept definition in conceptual design

Previously, the term **architecture** was defined as an ensemble of complex elements which together, have the ability to perform the mission. The fact that no simple atomic (un-dividable) solution can perform all functions together requires the integration of multiple solutions which together form the architecture. Each function can be performed through different solutions. An **architecture concept** corresponds to a specific combination of solutions. These solutions correspond to what we shall refer to as a **subsystem**. The architecture is there to accommodate these subsystems within an environment which will allow them to perform the mission together.

In order to do so, the architecture concept definition phase must provide information which can be classified under two categories [33]:

- Compositional: What are the elements composing the architecture?
- Structural: What are the relationships and interactions between the elements composing the architecture?

This section will be organized around these two aspects. First we will observe the ways available to guide and define the architecture composition. This aspect will require first the definition of the decomposition method necessary to the classification of alternatives. Based on this description we shall then focus on the composition definition methods. This description will then review methods facilitating the description of

structure. This section will be concluded by a comparison of the approaches described earlier.

3.2.1 **Decomposition and Composition Methods**

This section will first review the methods used to decompose the architecture. This decomposition is necessary to organize solution and create a space of alternatives. This review will be followed by the description of the methods associated with composition. Then, if decomposition allows for the creation of a space of alternatives, composition methods can be seen as a way to navigate in this space of alternatives, by guiding the architects and clearly pointing to a specific solution.

3.2.1.1 Decomposition Methods

Previously, the architecture concept was described as a specific combination of solutions at the subsystem level. In order to classify the possible combinations, a breakdown of the architecture into physical subsystems is necessary. The exploration of the combinatorial space allows the definition of architectural alternatives. But, in order for these alternatives to be coherent, a breakdown scheme must be used. This scheme defines the criteria by which the architecture (i.e. the aircraft) is subdivided into well defined and unambiguous subsystem categories. This scheme is what is defined as the **decomposition method**. In this section, four decomposition methods will be presented and analyzed. These methods are based on the following decomposition schemes:

- Disciplinary
- Physical
- Perimeter-based
- Functional
- ATA chapter

3.2.1.1.1 Disciplinary-Based Decomposition

The **disciplinary-based decomposition** approach groups ASA elements into their relevant disciplinary analysis groups (aerodynamics, structures, propulsion, control, electricity, pneumatics, hydraulics, data control, etc…). The "relevant" disciplines will be the one most likely to be impacted by the given system attributes.

In some situations, it may be difficult to relate specific systems to a single group (example: the wing is involved in aerodynamics, structures and control). Also in situations where the subsystem rely on different technologies (and therefore on different disciplines), the organization of subsystems among disciplines may become a challenge as the association between the disciplinary subsystem and the requirement implied by the mission changes. To illustrate this situation we may consider the electric and pneumatic subsystems. If we are considering an architecture including pneumatic deicing of the wing, the requirements related to deicing the wing are related to the pneumatic subsystem. If the architecture includes electric mats to deice, the deicing requirements are related to the electric subsystem.

Therefore, using a disciplinary based decomposition is appropriate in situations where there is a clear mapping between physical elements and discipline (i.e. no major subsystems are shared amongst disciplines) and where trades are not changing the association between the disciplinary subsystems and the requirements.

3.2.1.1.2 Physical Decomposition

The **physical decomposition** is certainly the most intuitive decomposition approach. A physical decomposition defines subsystems based on their coherent physical aspects. For example, a turbofan engine is composed of multiple elements (fan, compressor, combustor, turbine, control computer etc…). Since these elements are composing a single physical body, they are grouped into a single physical subsystem: the engine.

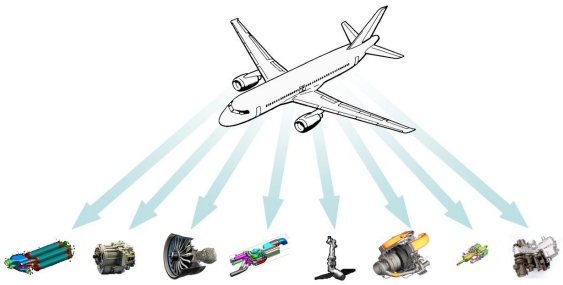


Figure 31: Physical breakdown

Physical decomposition is visual, therefore easy to understand. But, as different architectures are considered, the physical subsystems composing the architecture, their nature, and their role may change. Therefore, physical decompositions in an ASA design context are difficult to keep track of. Subsystem groups in a physical decomposition may look very different between alternatives, may have different roles in the architecture or may not even be needed in other alternatives. In a design context, where the decomposition is used as a foundation for defining new architectural concepts, having such an unstable framework to base the architecture conceptual definition does not simplify the process. But it is important to note that the boundaries of design experts are often defined by physical subsystems. The consideration of the physical subsystem is therefore essential to the sizing aspects of the analysis.

3.2.1.1.3 Geometric Decomposition

The **geometric decomposition** groups systems by the zone in which they are typically located. This approach assumes an installation scheme for the aircraft. For this reason, it facilitates the consideration of installation related factors on the sizing of the system. Some of these factors hold an important role in the sizing of the elements composing the architecture (e.g. thermal effects). On the other hand, breaking down the architecture based on where subsystems are typically installed, is not sufficiently precise or flexible in the definition of the architecture itself. This is due to the fact that the installation should be an architect design dimension, whereas using a perimeter based breakdown imposes a fixed installation scheme.

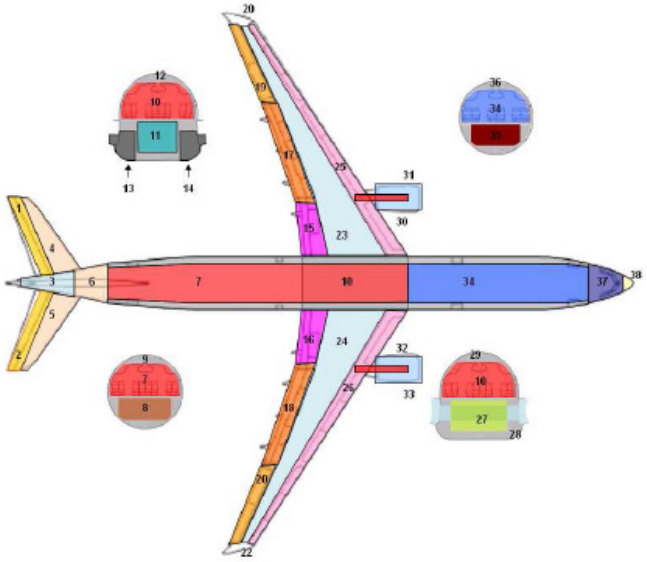


Figure 32: Notional geometric decomposition

3.2.1.1.4 Functional Decomposition

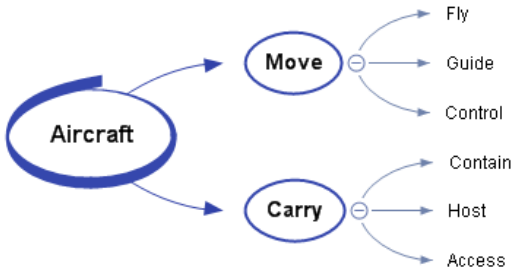


Figure 33: Notional functional breakdown[28]

The **functional decomposition** groups systems based on the function they fulfill within the architecture. In the context of a commercial aircraft, the functions to be fulfilled are not drastically changing from one design to the next. Therefore, unlike physical, geometrical and disciplinary elements which can be modified, traded or even eliminated, functional subsystem tend to be more persistent. Also the granularity of function decomposition can be placed at different levels based on the type of trades to be performed. The major disadvantage of a functional breakdown is the fact that it is less intuitive.

A function is an action. An action can not be represented by an object. Let's consider the function "provide light". A light bulb can be used to illustrate this function,

but the light bulb itself is a physical implementation of the function rather than the function itself. “Provide light” may refer in the same fashion to a candle, an incandescent material in general or anything else with the capability to “provide light”. Hence, in some cases functional decomposition may be less intuitive than other decomposition approaches as it is less explicit.

The fundamental advantage of a functional breakdown is the fact that it relates each subsystem to a function, therefore to a specific requirement. This decomposition method enables a clear link between the requirement formulation phase and the sizing activity necessary to the analysis of the architecture concept. Also since boundary functions are directly related to the requirements (which do not change with the architecture concept), functional requirements provide a stable breakdown for some elements of the architecture. But this stable breakdown does not extend to the full architecture. As pointed out in the previous section, some functions (induced functions) are not valid across architectural concepts. Therefore, when a functional decomposition is used to generate architectural concepts, a balance between completeness and universality of the decomposition must be identified.

3.2.1.1.5 ATA Chapter Decomposition

In addition to the four purist approaches, different decomposition schemes can be defined by combining them. The most popular and widespread hybrid approach is the **ATA chapter** breakdown. The ATA chapters were defined based on a finely tuned balance between the disciplinary, physical and functional approaches. These chapters were established in 1936 by the Air Transport Association (ATA) in order to simplify and harmonize maintenance practices [34].

Over the years, aircraft designs were systematically described and documented by ATA chapter. Hence, physical attributes like weights, costs, etc… were stored following

this outline. These databases became useful to generate historical regression for new developments. As a result, ATA chapters also became a standard for design purposes.

Figure 34: ATA chapters

There is, however, a significant shortcoming with ATA chapters. Due to the fact that the results from a mix of different approaches, as the architecture changes, the ATA chapter definition must be modified in order to classify new systems. This situation can be illustrated by the typical confusion resulting from the integration of a more electric version of chapter 21 (Air Conditioning). The label used for this chapter refers to a function: “condition the [cabin] air”. Indeed, a standard chapter 21 (e.g.: A330’s) will include only systems exclusively dedicated to this function and will not include the systems dedicated to the function of generating pneumatic power (located at the engine compressor). For a more electric chapter 21, the generation of pneumatic power (by the electric compressor) is directly connected to the air cycle machine and included in this chapter. In this case, the systems composing the chapter are no longer exclusively dedicated to the function of “conditioning the [cabin] air”. This reveals the fact that the ATA chapters are neither a purely functional approach nor a physical approach and as a result must be redefined for every new architectural concept.

This case by case redefinition of ATA chapters was quite acceptable for maintenance practices, namely once the architecture was already defined. Its clarity in terms of reference to specific physical subsystem, in specific locations, relevant to a specific discipline, made it appropriate for its original purpose. When it comes to supporting architecture development, the lack of a universal rule to classify systems among chapters adds difficulty to the design process.

3.2.1.1.6 Discussion on Breakdown Methods

The four fundamental decomposition methods are based on different perspective to the architecture. In all architectures, these four dimensions coexist and must all be considered at some point in preparation for the analysis of the concepts. Since functions relate the mission and the functional requirements to sizing of subsystems, the functional perspective facilitates the link between requirements and sizing. The physical perspective structures the sizing process. In a bottom-up sizing process, the physical elements constituting the architecture are sized based on their function within the architecture. The geometrical perspective is essential in characterizing the aircraft as a whole, its geometry and its weight distribution. As a result, it plays an important role in the aircraft synthesis activities. The disciplinary perspective is central to the synthesis activities (consolidation of the subsystem level to aircraft attributes) and contributes to the sizing of specific subsystems. These relationships described above are represented in graphical form in Figure 35.

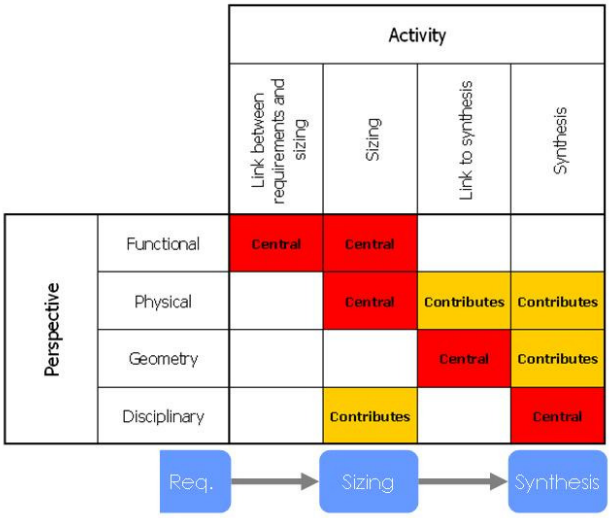


Figure 35: Role of architectural perspectives in the analysis activities

The most appropriate decomposition method will depend on the context of the study. When a perspective is chosen for breaking down the architecture, this dimension serves as a pivot for the generation of new concepts. Therefore, it is important that the pivot provides a stable breakdown which will enable the formulation of concepts without limiting the scope of the conceptual exploration. In the case of power architecture conceptual trades, the complexity resides primarily in the number of functional requirements (number of energy loads). Therefore, a functional breakdown is certainly preferable.

If we were to perform trades on a regional power grid management problem, we would be considering a fixed architecture concept (where the physical composition was fixed) but with changing functional requirements. In this type of trade a functional breakdown would not be appropriate and a physical or disciplinary breakdown would be preferable. Therefore we can conclude that there is no universal best solution to breaking down architectures. Among the decomposition approaches presented earlier, the most appropriate is the one offering a breakdown which will be the most consistent across architectures and will facilitate the preparation of the analysis.

3.2.1.2 Matrix-based Composition Methods

Now that we have observed the different ways available to us for breaking down architectures, we can proceed to the analysis of composition methods. In this review, composition methods are classified under two categories. The first corresponds to the matrix-based composition methods, the other to the language-based methods. This section will describe and discuss the first kind. The following will focus on the language-based.

3.2.1.2.1 Description of the concept of morphology

The matrix-based approach is based on the concept of morphology. This concept was initially proposed by the astronomer Fritz Zwicky [35]. It is based on a breakdown of the architecture. This breakdown defines categories of elements which constitute the architecture. In each of these categories, several alternatives may be selected. The concept of morphology is based on the classification of these alternatives. This classification facilitates the identification of possible concepts by exploring the combinatorial space offered by the classification.

The tool which resulted from this concept is the **morphological matrix** (or **matrix of alternatives**). The morph matrix lists on its rows the categories identified from the breakdown of the architecture. The cells on each row constitute a list of alternatives available for each category.

		Alternative			
Function	**Condition air**	Electrically compressed air	**Engine bleed air**	Engine bleed + electric booster	
	De-ice wings	**Pneumatic concept**	Electro-thermal concept	Electro-impulse concept	Hybrid concept
	Control flight	Electro-mec. Actuator concept	Electro-hydrostatic actuator concept	**Hydraulic actuator concept**	Hybrid EHA/EMA/HA concept
	Fly	**Conventional wing**	Blended wing	Canard	
	Protect passenger and payload	Aluminum	**Composite**		
	Propel	PW6000	T1000	**CFM56**	

Figure 36: Notional Matrix of Alternatives

The matrix of alternative facilitates the identification of possible architecture compositions. The user points to one cell on each row to identify an alternative for each subsystem in the architecture. Figure 36 presents an example of a matrix of alternatives. This matrix is based on a functional decomposition. Therefore, each row is composed of the alternative candidates for that function. The red cells represent alternatives which were chosen among the alternatives. So the matrix as shown in Figure 36 represents one architecture composition.

3.2.1.2.2 Interactive Composition Matrices

The disadvantage of the matrix of alternatives is the fact that among the combinations of architecture composition, some are either unrealistic or obviously suboptimal. This disadvantage motivated the development of the **Interactive Reconfigurable Matrix of Alternatives** (IRMA) [36]. This tool associates to the basic morphology concept, the means to record the tacit knowledge about the relationships between the choices on each row. It also allows applying filters to the alternatives, based on user defined critera (e.g. Technology Readiness Level). The following figure represents an example of an IRMA presented by Engler in [36]

Cruise Speed	Subsonic		Supersonic		Hypersonic		Orbital	
Engine Type	Turbofan		Turbojet		Ramjet		Turboramjet	
	Pulse Detonation		Combined Cycle		Other			
Number of Engines	1		2		4		Other	
Ferry Range	<1000 nm		1000-3000nm		3000-5000 nm		>5000 nm	
Refuelable	Yes		No					
Piloting	Manned		Unmanned/Remote		Unmanned/Autonomous			
Stores	External		Internal Exposed		Internal Enclosed			
Wing Morphing	None		Variable Sweep		Variable Camber			
Body Style	Blended Wing		Flying Wing		Conventional			

Figure 37: IRMA exemple [Engler et al 2007]

The example above illustrates the capability of the IRMA to filter out incompatible alternatives. This IRMA was designed for concept exploration for a missile launcher aircraft. In this matrix, a ramjet concept was selected for propulsion. Since the

operational envelope of the ramjet is limited to supersonic flight, all other alternatives were eliminated in the “cruise speed” row.

In order to register the incompatibilities, the IRMA is built on an incompatibility matrix. A portion of the compatibility matrix, corresponding to this IRMA is presented in Figure 38. It shows the compatibilities between the cruise speed and propulsion alternatives.

		Cruise Speed				Engine Type							
		Subsonic	Supersonic	Hypersonic	Orbital	Turbofan	Turbojet	Ramjet	Turboramjet	Scramjet	Pulse Detonation	Combined Cycle	Other
Cruise Speed	Subsonic	3	1	1	1	0	0	1	1	1	0	0	0
	Supersonic	1	3	1	1	0	0	0	0	1	0	0	0
	Hypersonic	1	1	3	1	1	1	1	1	0	0	0	0
	Orbital	1	1	1	3	1	1	1	1	1	2	1	1
Engine Type	Turbofan	0	0	1	1	3	1	1	1	1	1	1	1
	Turbojet	0	0	1	1	1	3	1	1	1	1	1	1
	Ramjet	1	0	1	1	1	1	3	1	1	1	1	1
	Turboramjet	1	0	1	1	1	1	1	3	1	1	1	1
	Scramjet	1	1	0	1	1	1	1	1	3	1	1	1
	Pulse Detonation	0	0	0	2	1	1	1	1	1	3	1	1
	Combined Cycle	0	0	0	1	1	1	1	1	1	1	3	1
	Other	0	0	0	1	1	1	1	1	1	1	1	3

Figure 38: Compatibility matrix

The compatibility matrix has as many rows and columns as there are subsystem alternatives listed in the IRMA. The compatibility matrix can take three values:

- 0: implies that the alternatives are neutral with respect to each other
- 1: implies that the alternatives are mutually exclusive
- 2: implies that if the alternative on the row is selected the one on the column is automatically selected.
- 3: the row and the column correspond to the same alternative.

3.2.1.2.3 Conclusion and discussion about the concept of morphology

The matrix of alternatives guides the definition of the architecture by organizing the alternatives and highlighting the possible combinations of subsystem alternatives which can constitute an architecture composition. Even with limited expert knowledge the user can define the composition of a broad variety of architecture concepts. But composition alone is not sufficient to the definition of the concept. The composition by definition specifies what is in the architecture, but it is not explicit on how these subsystems are interrelated. This limitation leads us to the next aspect of the architecture definition which is structural definition.

3.2.1.3 Language-based Composition Methods

The need for clear and unambiguous description methods for complex architectures has been a major concern in the industry over the past few decades. This need has motivated the computer science community to define a graphical language that can be used to describe software architecture. This effort resulted in the creation of the **Unified Modeling Language** (UML) [37]. The success of this modeling language has fostered interest in the system engineering community which used it as a basis for a new modeling language dedicated to the description of system architectures. These developments resulted in the creation of the System Modeling Language (SysML)[38-39].

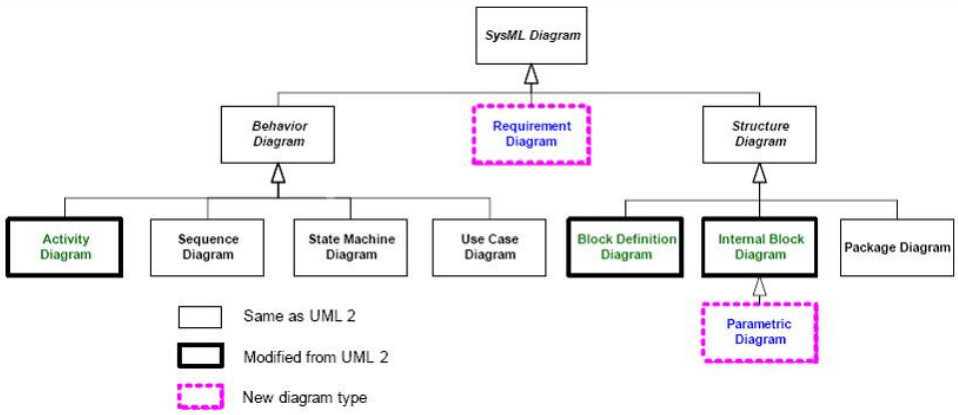


Figure 39: SysML diagram

In the same fashion as one can define an architecture by describing it with sentences in English and/or back-of-the-envelope graphs, SysML portraits this architecture using standardized semantics and notations. SysML is structured around three categories of diagrams presented in Figure 39. The physical description of the architecture is supported by the "structure diagrams". In order to describe the composition of the architecture, the Internal Block Definition diagram is of particular interest to us. This type of diagram allows for the specification of the composition of an architecture by using the containment relationship symbol. This symbol is described by a line connecting the superstructure (element containing) to the element contained. The line on the superstructure-side is indicated by a black diamond while the contained element is designated by an arrow [40].

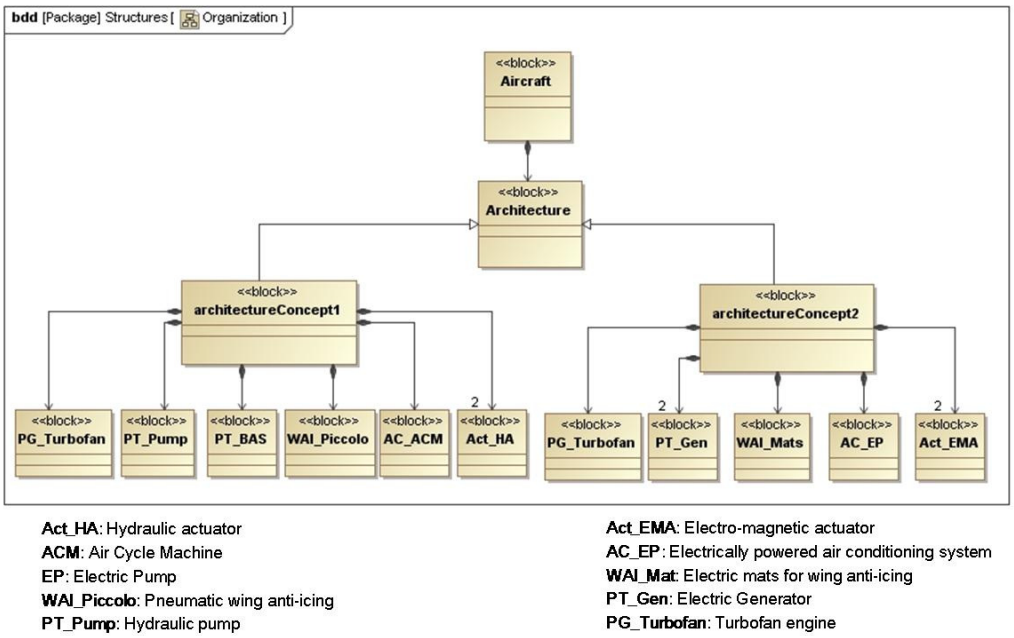


Figure 40: Block definition diagram example

Figure 40 shows an IBD presenting two architecture concepts and their composition. The white triangular arrows represent the fact that "architectureConcept1" and "architectureConcept2" are two different versions of the entity "architecture". The composition of each concept is defined by the black diamond shape arrows connecting the concept to its subsystems.

3.2.2 **Architecture Structural Definition Methods**

Defining what the architecture contains does not provide a complete description of what the architecture is. The role of the subsystems with respect to each other must be defined. The functional structure describing these mutual relationships defines what we shall refer to as the **architecture structure**.

This section describes several methods that can be used to describe the structure of architectures. The methods which will be described will start from the most simple and visual, to the most abstract and precise. The methods that will be discussed will be the following:

- Graphs
- Structure matrices
- Genetic network strings
- Internal block diagrams

Before we move on to the description of the various techniques, let us consider a baseline architecture which will be used in this section to compare the different methods. This architecture is composed of:

- Two hydraulic actuators
- An Air Conditioning unit (AC) consuming pneumatic energy
- A Wing Anti-Ice system (WAI) consuming pneumatic energy
- One hydraulic pump
- Pneumatic distribution system (or Bleed Air System - BAS)
- One engine

The architecture is structured as follows:

- Both actuators are connected to the pump for hydraulic energy
- The AC and WAI are connected to the generator for electrical energy
- The pump and the generator are connected to the engine for mechanical energy

3.2.2.1 Interaction Graphs and Matrices

Interaction graphs and matrices are common techniques for architecture description. A detailed description of these techniques is available in chapter 2 of reference [41].

The architecture presented above can be represented as a graph. The first type of graph (represented in Figure 41) is an **interaction graph**. The interaction graph displays which elements are in contact. In this representation, the relationships are undirected. Therefore we can see which elements are in contact with one another but we do not know their mutual role in the relationship. (Note: the colors in this graph are only informative to highlight the different type of subsystems).

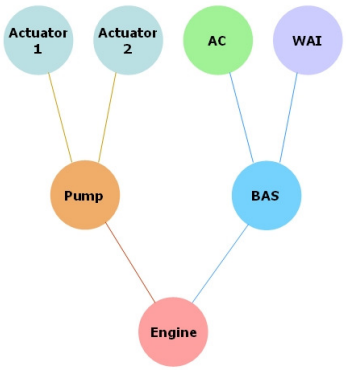


Figure 41: Interaction graph

The interaction graph can be represented in a mathematical form with a **self interaction matrix** (represented in Figure 42).

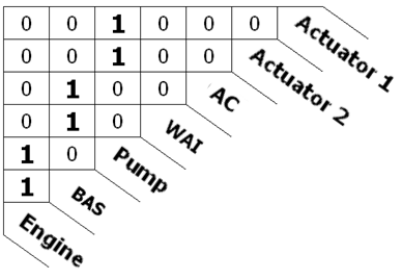


Figure 42: Self-interaction matrix

In a sizing process, each subsystem receives requirements. These requirements drive the sizing process. In a relationship between two elements (e.g. actuator 1 and

pump), one of the element (actuator 1) is imposing a requirement on the other (pump). Therefore, in sizing relationships the connections are always directed. In order to capture the orientation in the relationship, **directed graphs** can be used.

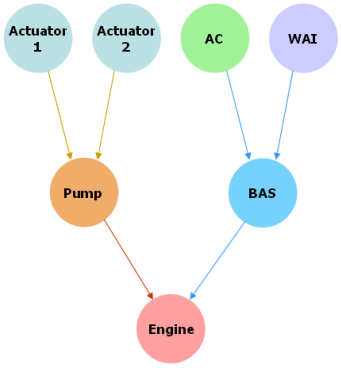


Figure 43: Directed graph

The directed graph can be described mathematically using a (directed) **interaction matrix**. The interaction matrix corresponding to the architecture example structure is shown below.

	Act. 1	Act. 2	AC	WAI	Pump	BAS	Engine
Act. 1	0	0	0	0	**1**	0	0
Act. 2	0	0	0	0	**1**	0	0
AC	0	0	0	0	0	**1**	0
WAI	0	0	0	0	0	**1**	0
Pump	0	0	0	0	0	0	**1**
BAS	0	0	0	0	0	0	**1**
Engine	0	0	0	0	0	0	0

Figure 44: Directed interaction matrix

This matrix is n by n in size where n correspond to the number of elements present in the architecture. Value stored in the matrix describes the relationship between the elements referenced by the row and the column:

- If aij = 1 then there is a relationship going from the element represented by row i toward the elements in column j.

- If aij = 0 there are no relationship going from I to j.

The graph representation provides a visual and effective way to represent the structure of the architecture. The interaction matrix provides a simple and mathematical mean to describe the architecture structure. This mathematical formulation offers the opportunity for automated processing of the architecture.

We can also observe that the interaction matrix is somewhat similar to the matrix of alternatives. If we assume that each row the matrix in Figure 44 represents a functional requirement, the alternatives listed along the row could be considered as physical alternatives to perform this function.

But if the interaction graph and matrices indicate the presence of relationships between the elements, it does not clearly indicate the nature of the relationship. In order to accommodate this lack of structure the methods presented in the following paragraph were defined to define the relationships in more detail.

3.2.2.2 Genetic Network Strings

The Genetic Network Strings (GNS) formulation was proposed by Bjorn Cole [42-43]. It is based on techniques which originated in the field of genetic network programming [44]. This network formulation is described by strings of variables. This formulation was motivated and shaped by two needs. The first is that the formulation needed to have a mathematical (or more exactly a symbolic meaning) which could be used by an optimizer in order to automate the exploration of networking concepts. The second is that the information stored in the string must provide sufficient details in order to automatically setup the Input/Output relationships of models representing the subsystem composing the architecture.

In order to understand the formulation of the genetic network formulation, some of the numerical aspects beyond the concept should be considered. So before describing

the formulation let's considered a little further the type of information necessary to size the architecture in the example.

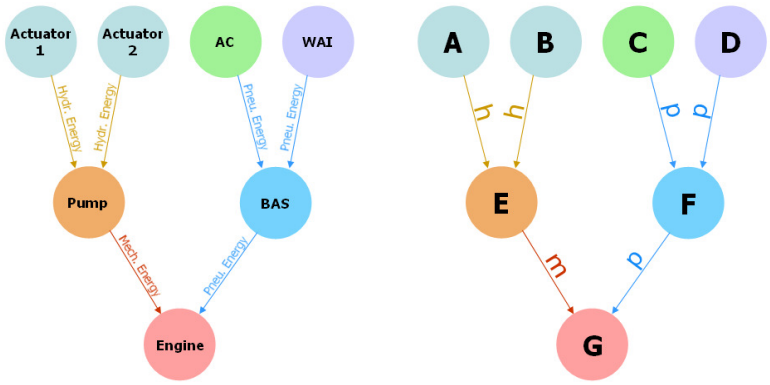


Figure 45: Networks with explicit connections

With this approach the models and the connections are classified by type. Each type of model corresponds to a different sizing analysis. These types correspond to "actuator" (in clear blue), "AC" (in green), "WAI" (in purple), "Pump" (in orange), "BAS" (in blue) and "Engine" (in red). Each type of model will have different interfaces. The model types and their interfaces are represented in Figure 46.

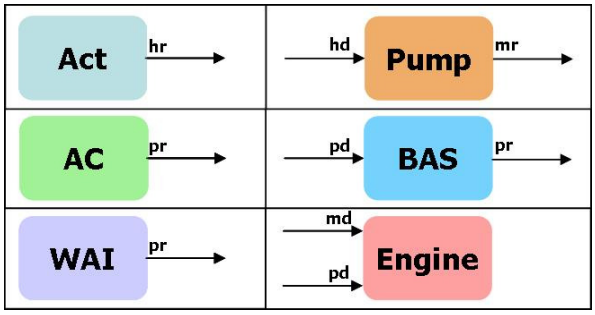


Figure 46: Models' I/O ports

Each model is expecting a certain type of information to be input in their interfaces. If we consider the pump model, we can see that it has 2 interfaces. One interface corresponds to the hydraulic energy demanded by its users (hd) and the other on to the mechanical energy required by the pump to operate (mr). We can see that the engine as two inputs (md and pd).

The connections between models are classified by the type of information they transfer. There are three types of connection in this problem: hydraulic energy, pneumatic energy and mechanical energy.

When using the GNS, the architecture is portrayed a series of string. The string is composed of multiple lines; each line corresponds to a specific node and the connection originating from it. The structure of the line is defined as follows:

		Outgoing connection 1	Outgoing connection 2		Outgoing connection *n*
Name of the node	Type of the node	. . . .	. . . .	... ...	. . . .

The four elements characterizing the sub-string for each outgoing connection is defined as:

Name of destination	identity of output interface	identity of input interface	type of the connection

Using these conventions the network definition of the example architecture is:

A	Act	E	hr	hd	h
B	Act	E	hr	hd	h
C	AC	F	pr	pd	p
D	WAI	F	pr	pd	p
E	Pu	G	mr	md	m
F	BAS	G	pr	pd	p
G	Eng				

Figure 47: Genetic Network String of the example architecture

We can observe that the GNS implies information which was not captured by the matrix-based methods. It can also be concluded that GNS includes the information necessary to describe most architectures. But if the GNS is a good means for qualifying the relationships between the subsystems, it certainly does not provide the means to describe a structure visually or to create a new one manually.

3.2.2.3 Descriptive Languages

In order to capture any arbitrary complex architecture we must turn toward the SysML language. In SysML, the structure of the architecture is represented by the internal block diagram (IBD). The IBD presents the exchange between the elements composing the architecture. The figure below represents the same architecture as the one described previously.

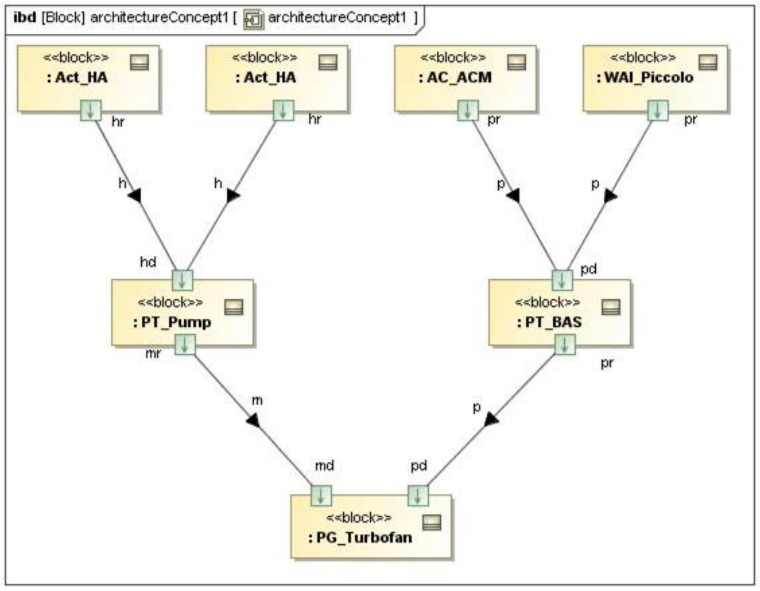


Figure 48: Internal block diagram example

It is important to note that the IBD captures many levels of detail which can potentially be translated into an analysis model structure. The port semantics provided by the language allow for the identification of the specific interface by which a subsystem is interacting with its environment. The ports are indicated in the figure above by the green arrowed elements on the boundary of boxes representing the subsystems. The type relationship can be qualified using conveyed flows. Conveyed flows are indicated by line with a black arrow. The characters next to the arrow indicate the type of the relationship.

3.2.3 **Comparison of Architecture Definition Methods**

At this point we have introduced several techniques supporting the definition of the architecture. In order to identify the most suitable orientation for the methodologies to be proposed in this thesis, it is necessary to compare the performance of the techniques presented earlier. In this section, a critical comparison is proposed. First the criteria used in the comparison will be introduced and discussed. Then, a simple test case will be introduced to benchmark the alternatives and illustrate their performance. The implementation of this test case will be followed by concluding remarks on the best approach.

3.2.3.1 Selection of Criteria for Comparing Definition Methods

The criteria used to compare the definition methods are introduced in two sections. The first section will discuss the technical requirements implied by the definition of an architecture concept. The second will discuss the non-technical requirements related to the organization of architecture trade-offs and their industrial context.

3.2.3.1.1 Technical Requirements

The solution desired solution must be able to capture alternatives in a way that guides the definition process. The definition of the architecture was broken down into two main parts: The composition and the structure. Each of these parts implies capturing different elements of the architecture definition. In addition to the definition, we shall also consider how the different approaches can be used to guide the definition of the architecture. Their capabilities will be compared by looking at the organization of the information and its potential to guide the model.

3.2.3.1.2 Non-Technical Requirements

Beyond the technical role of architecture definition, it is important to consider its context. The fundamental motivation for this activity can be found in the formulation of

Objective 3 and *4*. The notion of acceleration in *Objective 3* (acceleration of architecture analysis) implies that the architecture definition method must provide a practical way to explore possible alternatives. The ability of capturing possible alternatives is also aligned with O*bjective 4* (detection of architectural opportunities). In the previous chapter, we have observed that aircraft system architectures are highly complex in their nature. Their constitution implies very diverse technologies; hence they require a broad scope of expertise in their definition and analysis. Therefore, the definition of aircraft system architectures is never a one person job. Based on this observation it must be ensured that the method for defining architecture concepts is suitable for collaborative developments. This suitability will be appreciated on the following aspects:

- Visually explicit description of architecture
- Simplicity of description methods

The explicitness of the method is necessary to unsure that it can be easily understood unambiguously by a large panel of experts with diverse perspectives. This aspect is necessary both from a collaborative and an effectiveness point of view. Since the architecture definition is also used to define the analysis model, an ambiguous or unclear concept definition method increases the difficulty associated with debugging the model.

The simplicity of the method is necessary to successfully implement this methodology in an industrial context. If the method is complex, it will require more time to train participants. This longer training period is done at the expense of actually performing the architectural trades. Therefore, the more complex this method, the less effective it will be.

3.2.3.2 Test Case Application for the Experimentation

In this experiment we will compare the ability to capture alternatives for a simple architecture expected to provide hydraulic power using shaft power from an engine. The

possible subsystem alternatives include electric pumps, mechanical pumps and electric generators. In order to designate architecture concepts the symbols presented in Figure 49 are used. A sample architecture is presented in Figure 50. It presents an architecture where the hydraulic power is provided via an electric pump, itself energized by an electric generator. The generator is connected to the engine.

Figure 49: Informal symbols for subsystems and relationships

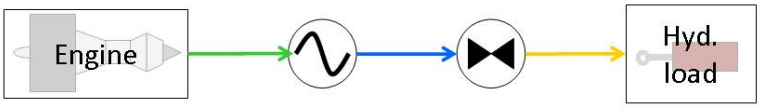


Figure 50: Informal representation of concept 0

Note the architecture considered in this test case will only include the elements necessary to the transformation of the mechanical power into hydraulic power. Therefore, in the architecture presented in Figure 50 only the generator and pump are part of the architecture. On the other hand, the engine and actuator are interfacing with the architecture but are not part of it.

In order to observe the ability of the various methods to capture specific alternatives the following four concepts are defined:

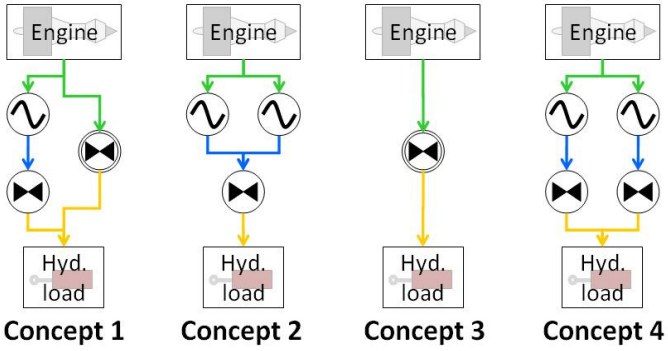


Figure 51: Notional concepts for test case

3.2.3.3 Deployment of Architecture Definition Methods

The description of the implementation on the test case will follow the same organization as the review. First the implementation of the architecture breakdown will be considered, then we shall consider the composition definition, to finally conclude with the structure

3.2.3.3.1 Decomposition of the Test Case

In this test case, the possible architecture alternatives can be considered from two perspectives: functional and physical. The functions that are physically present in this architecture are the provision of mechanical and electric power. The provision of hydraulic power can be classified as a **boundary function** (function directly implied by the mission). The other functions will be implied by the chosen subsystems and are therefore classified as **induced functions**.

Each of these functions can be implemented by different physical alternatives. Three alternatives were listed in Figure 49. Each of these alternatives can be integrated an arbitrary number of time. It is possible to have architectures with no generator, or with one, two or three (…) generators.

If it is clear that both functional and physical alternatives are both relevant perspectives on this architecture, none of them alone provide a satisfactory breakdown. If we consider the functional breakdown, the electric function is not permanently present in the architecture. The physical breakdown is even less relevant as since none of the physical subsystems will be systematically present in the architecture.

3.2.3.3.2 Formulation of Compositional Alternatives

If we were to consider the formulation of composition alternatives with a matrix of alternatives, two options would be available to us. The first option would be to list alternatives using a physical breakdown, the other using a functional breakdown.

Table 1: Physics-based MA

Elec. Pump (EP)	0	1	2	3	…
Mech. Pump (MP)	0	1	2	3	…
Generator (Gen)	0	1	2	3	…

Table 2: Functional-based MA

Prov. Hyd. Power	1 EP	1 MP	1MP/1EP	1MP/2EP	…
Prov. Elec. Power	None	1 Gen	2 Gen	3 Gen	….

If we are using a physics-based Matrix of Alternatives (MA), any combination including at least one mechanical pump will be valid. The alternatives using one or more electric pumps will be valid if and only if, at least one generator is included. Similarly, for the functional-based MA, all combinations including a generator without an electric pump (or vice versa) will not be valid. Using an IRMA, it is possible to address these incompatibilities. But implementing the compatibility matrix can become a fairly tedious task for larger and more complex architectures. The obvious advantage of this approach is its ability to parameterize the alternatives. In a simple but open-ended design space such as this test case, the matrix-based approach allows an easy parameterization of the design space.

The matrix of alternatives can capture the composition of the four concepts. Their descriptions are presented in the following tables:

Table 3: MA implementation of concept 1

Elec. Pump (EP)	0	1	2	3	…
Mech. Pump (MP)	0	1	2	3	…
Generator (Gen)	0	1	2	3	…

Prov. Hyd. Power	1 EP	1 MP	1MP/1EP	1MP/2EP	…
Prov. Elec. Power	None	1 Gen	2 Gen	3 Gen	….

Table 4: MA implementation of concept 2

Elec. Pump (EP)	0	1	2	3	…
Mech. Pump (MP)	0	1	2	3	…
Generator (Gen)	0	1	2	3	…

Prov. Hyd. Power	1 EP	1 MP	1MP/1EP	1MP/2EP	2EP
Prov. Elec. Power	None	1 Gen	2 Gen	3 Gen	….

Table 5: MA implementation of concept 3

Elec. Pump (EP)	0	1	2	3	…
Mech. Pump (MP)	0	1	2	3	…
Generator (Gen)	0	1	2	3	…

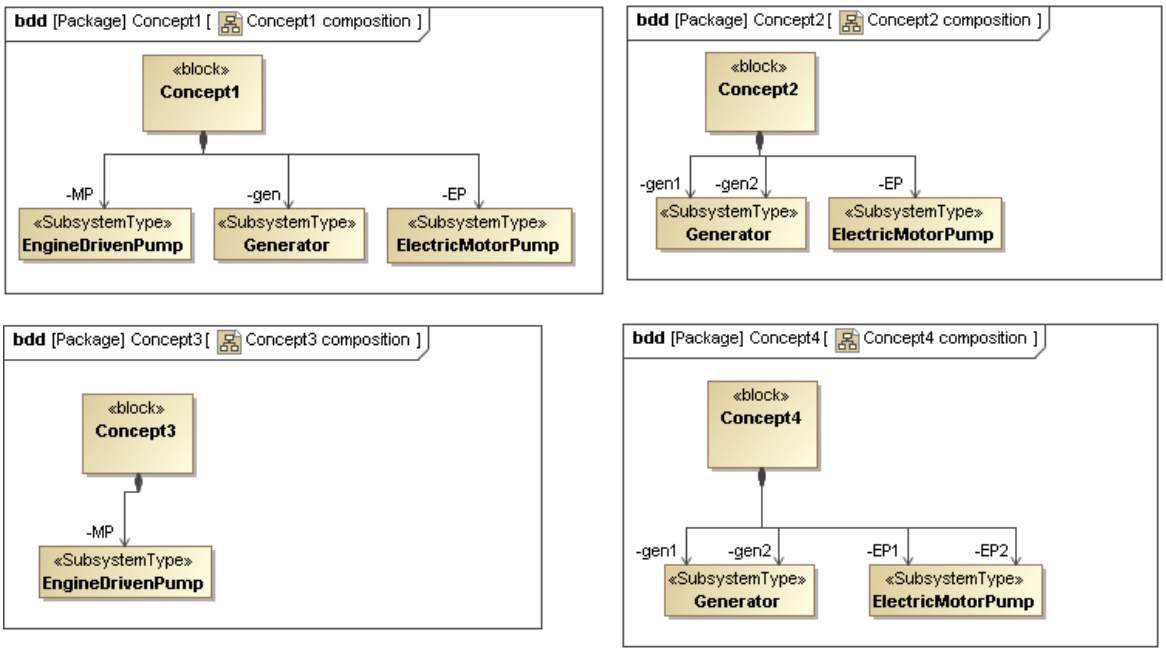
Prov. Hyd. Power	1 EP	1 MP	1MP/1EP	1MP/2EP	…
Prov. Elec. Power	None	1 Gen	2 Gen	3 Gen	….

Table 6: MA implementation of concept 4

Elec. Pump (EP)	0	1	2	3	…
Mech. Pump (MP)	0	1	2	3	…
Generator (Gen)	0	1	2	3	…

Prov. Hyd. Power	1 EP	1 MP	1MP/1EP	1MP/2EP	2EP
Prov. Elec. Power	None	1 Gen	2 Gen	3 Gen	….

The SysML based approach would define the composition using block definition diagrams. The four concepts are presented in the following figure. We can see implicitly that the definition of the composition is defined purely on a physical breakdown. There are no indications of the functionalities of the constituting subsystems in this diagram.



It is also important to note that SysML only allows for the definition of concepts and, unlike the Matrix of Alternatives, it provides no indication of the ensemble of possible alternatives. However it enables the differentiation of the elements by assigning a specific name to each subsystem.

3.2.3.3.3 Formulation of Structural Alternatives

The matrix of alternatives does not support directly the definition of the structure. The elements selected in the matrix can not be related to each other. For instance in concept 4 it is not possible to relate which of the two generators supports which electric pump. In order to capture the architecture definition, we shall assume that the selections in the matrix of alternatives construct the rows and columns in a directed interaction matrix. Using this assumption the following matrix would be used to define the test concepts. In these matrices, the impossible connections were grayed out. The feasibility of the connection indicated by each cell can be determined by looking at the function performed by the item listed on the column and the function required on the row. If the functions match the connection is feasible.

Table 7: Structure concept 1

	Prov hydr. Power	Gen	MP	EP
Prov. Shaft Power		⊙	⊙	
Gen				⊙
MP	⊙			
EP	⊙			

Table 8: Structure concept 2

	Prov hydr. Power	Gen1	Gen2	EP
Prov. Shaft Power		⊙	⊙	
Gen1				⊙
Gen2				⊙
EP	⊙			

Table 9: Structure concept 3

	Prov hydr. Power	MP
Prov. Shaft Power		⊙
MP	⊙	

Table 10: Structure concept 4

	Prov hydr. Power	Gen1	Gen2	EP1	EP2
Prov. Shaft Power		⊙	⊙		
Gen1				⊙	
Gen2					⊙
EP1	⊙				
EP2	⊙				

We can see that the structure matrix provides a good overview of the feasible structures given an architecture composition. For instance the only concept which offered multiple structure alternatives was concept 4. Although it provides a vision over the possible structures, it does not provide an intuitive vision of the selected concept. The matrix and the tokens indicating the connections offer only a cryptic representation of the structure which is not necessarily trivial to understand.

If we use the internal block diagram to represent the structure we can see that the structure can be expressed in a more intuitive fashion. This diagram also provides significantly more information concerning the type of the connections. However, it does not highlight the possible alternatives.

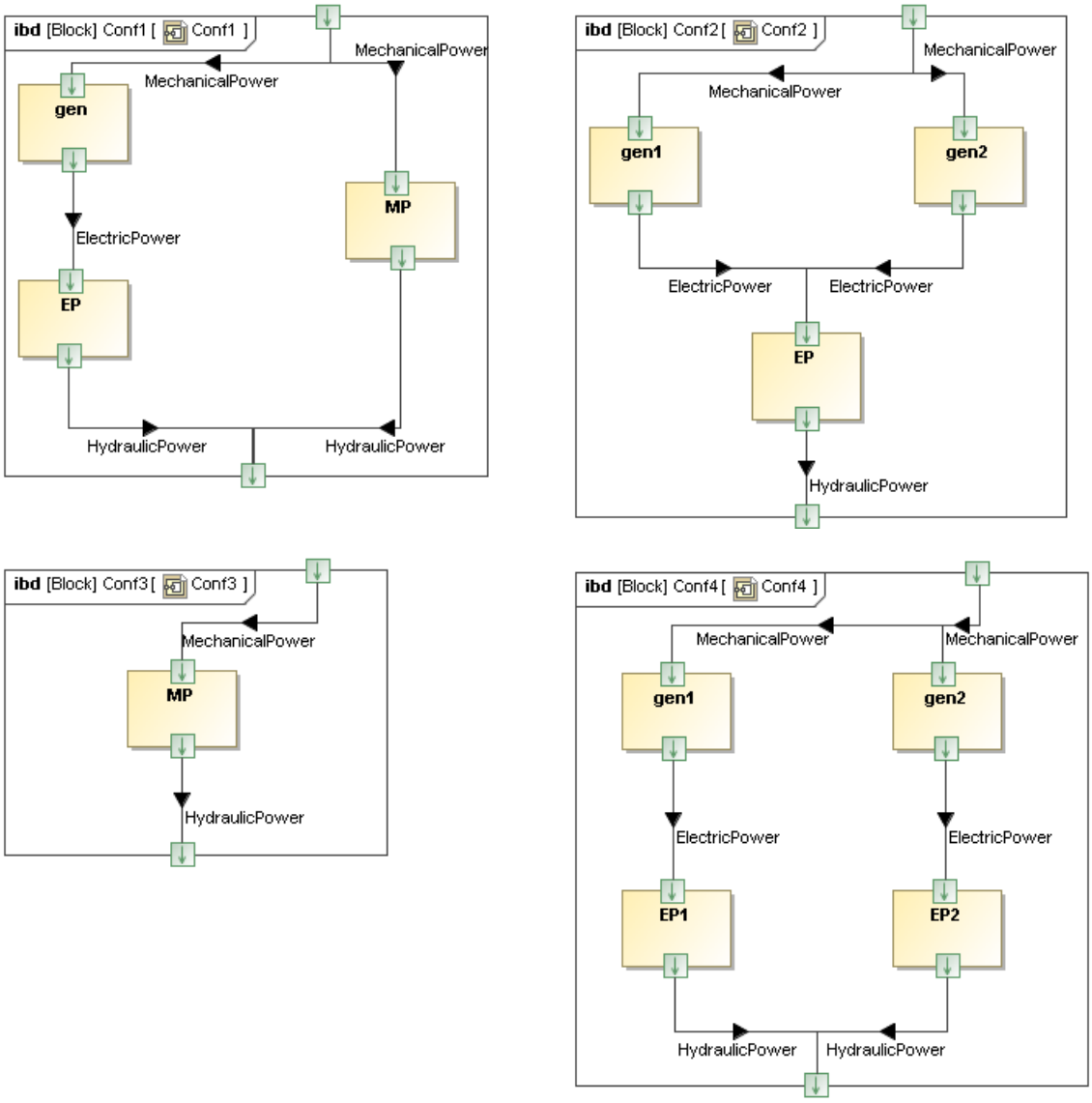


Figure 52: SysML description of architecture structure

3.2.3.4 Conclusions about Architecture Conceptual Definition

In the comparison presented previously, we have observed that the matrix-based solutions offer a good means to parameterize the conceptual design space. But several important aspects limit their application to the problem at hand. The first important limitation is the fact that in order to successfully define reasonable conceptual alternatives, a significant amount of logic must be captured. This challenge can be traced back to what we shall refer to as **architectural depth**. Architecture depth characterizes

the fact that some boundary function can not be directly performed by a single subsystem (for instance “provide hydraulic power” in the test case above), the definition requires a deeper level of functionality. The second level of functionality defines the induced functions which come and go depending on the physical implementation. Since the matrixes of alternatives require a fixed breakdown to classify alternatives, they do not naturally capture this dimension. In reference [33, 45] Armstrong et al attempted to address this limitation by introducing the Adaptive Reconfigurable Matrix of alternative (ARM). This solution which is discussed in further details in Appendix XXX, redefines a matrix of alternative every time a new selection is made. Although the ARM addressed the limitation associated with the lack of ability to capture architecture with depth associated with typical matrix-based methods, it did not address their visual limitation. Using a matrix to describe an architecture concept does not clearly and unambiguously express the architecture concept.

On the other hand the SysML approach allowed for a visually intuitive description of the architecture which can be used to by a large group of people. Architecture description solutions based on SysML were also developed by software vendors. These solutions provide a mature and deployment-ready solution for the description of architectures in an industrial environment. This advantage is to be considered also in the context of some important difficulties associated with the use of a SysML-based approach.

The first important limitation of SysML is its lack of ability to guide the architecture definition. In the context of the matrix-based approaches, decisions toward the description of the architecture can be guided step by step (each line in the matrix corresponding to a decision). Since the solution developed in this thesis is oriented toward an Integrated Product Team of experts, we shall assume that this functionality is not essential. Also one can argue that if the subsystems are defined both physically and

functionally, rules can be put in place to, if not guide the definition, guard against unfeasible or incoherent decisions.

The other important limitation of SysML is the complexity of its rhetoric. SysML is a modeling language designed for a very large scope of applications in mind. Although its universality is an important feature of SysML, the complexity it implies also constitute somewhat of a barrier against its popularization as a design language. Based on the observation presented earlier, only a small subset of the SysML diagrams are necessary to the basic description of aircraft power system architectures. By focusing on few and conceptually simple elements of the language, the complexity associated with the use of SysML can be alleviated.

Table 11 provides a qualitative overview of the conclusions provided by the comparison between matrix based methods. This table provides us with a visual summary of observations drawn previously.

The matrix-based solutions (Matrix of alternatives and design structure matrix) provide a format which facilitates the parametric definition of architectures. In a context where we try to perform a machine based exploration (or optimization) of the architecture concepts, the matrix-based techniques provide a valuable definition environment. But previously we also observed that matrix–based solutions are not a reliable solution in capturing architecture alternatives with different functional breakdowns. A significant amount of work was dedicated to the reduction of the effect of this limitation [33, 45-48]. But these solutions remain highly impractical from a collaborative point view and limited in their scope. The other important limitation of matrix based methods is the fact that they are not visual. In the context of the framework proposed in this thesis, the lack of an intuitive and visual tool would be an impractical limitation.

Table 11: Comparison overview of Matrix/SysML based methods

		Matrix-based definition		SysML-based definition	
		Matrix of alternative	Interaction Matrix	Block Definition Diagram	Internal Block Definition diagram
Definition of the composition	Ability of listing alternatives	Good		Poor	
	Conceptual clarity of described concept	Neutral		Good	
	Visual clarity of described concept	Bad		Good	
Definition of the composition	Ability of listing alternatives		Good		Poor
	Conceptual clarity of described concept		Neutral		Good
	Visual clarity of described concept		Bad		Good
Conceptual simplicity of method		Good	Good	Poor	Poor

The alternatives to matrix solutions were the graph-based solutions. These solutions included oriented graphs and the System Modeling Language graphs. The major advantage of these methods was the fact that they were significantly more explicit and visual than the matrix-based solutions. This advantage facilitates collaboration within the architecting team and will make the solution easier to correct in the situation of large and sophisticated architecture concepts. As a result, the most important feature necessary from the architecture definition environment is its ability to support collaborative work. SysML is a language which was defined for multidisciplinary collaboration and complex system definition. Also the SysML graphs can be saved in class structures (MagicDraw ® or ARTiSAN ®). Based on this feature, it is possible to describe architecture concepts in a format both visually intuitive and accessible by a machine (hence enabling the automated translation from concept to model).

3.3 Modeling and Analysis Approaches

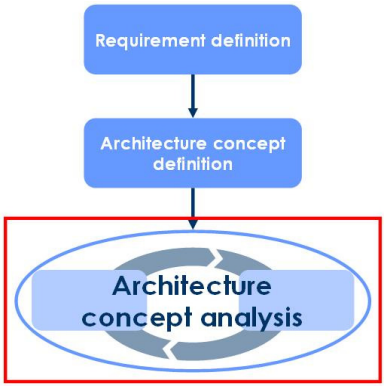


Figure 53: Role of modeling and analysis in conceptual design

Both in the academic literature and in industrial practices, there are a broad variety of approaches to the analysis of architecture concepts. In this section, the review of these approaches is broken down into two sections. The first section will describe how expert knowledge can be integrated in a fashion that facilitates numerical analysis of the architecture. The second will describe different approaches to architecture analysis.

3.3.1 **Expert knowledge integration approaches**

The expert knowledge integration approaches were broken down in two types. The first type will qualify typical system engineering approaches and will be described through the quality function deployment approach. The other type is the Multi-disciplinary approach. This approach focuses more on mathematical solutions supporting integration of technical analyses.

3.3.1.1 Quality Function Deployment

The **Quality Function Deployment** (QFD) is a method for understanding the relationships between technical parameters and customer requirements. This method is constructed around a tool called the House of Quality also referred to as QFD matrix. This matrix is in fact a collection of several matrices which together link engineering parameters to the Figure of Merit of the design problem. [49]

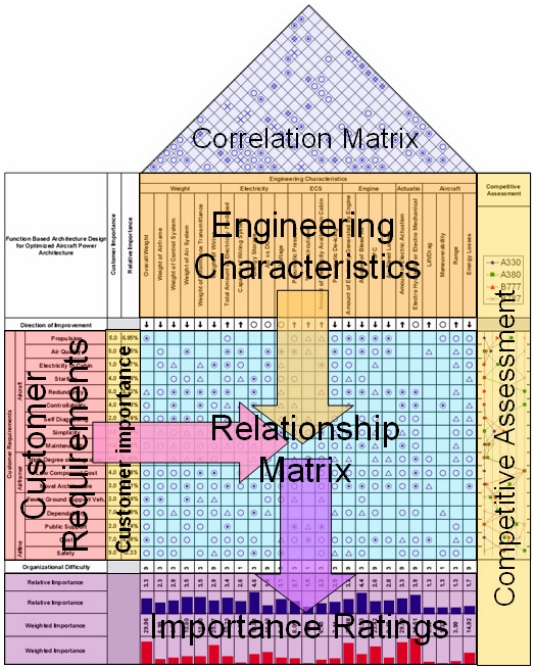


Figure 54: Elements of the QFD[28]

The matrix is structured around two interdependent parameters:

- Engineering characteristics
- Customer requirements
- Quality of the solution

The engineering characteristics refer to parameters controlled by design decisions. These engineering characteristics refer to things like technology characteristics or design parameters. The customer requirements refer to attributes of the solution. These attributes are figures of merit qualifying the performance of design on some specific customer requirements. The quality of the solution is not present explicitly in the QFD, but the outcome of the analysis of the QFD will implicitly suggest design decisions optimizing the overall quality of the design. In order to understand the true meaning of each element in the matrix, let us consider their mathematical meaning. In this analysis, we shall refer to engineering characteristics as *Xi*, the Customer requirements (or system attributes) as *Ai,* and the quality of the solution (or overall utility) *U*.

The central element of the QFD is its relationship matrix. This matrix defines how the performance on customer requirement changes depending on the performance on engineering characteristics. The relationship matrix is of size *n* by *m*, where *n* is the number of customer requirements and *m* the number of engineering characteristics. The cell (i,j) in the relationship matrix is defined as:

$$RM_{i,j} = \frac{\partial A_i}{\partial X_j} \tag{6}$$

The customer importance ranking is a table of size *n* by 1 (where n is the number of customer requirements). It defines a ranking which relates the performance on a specific customer requirement to the overall quality (or utility) of the design. The rankings are used to define weighting factors which will define the relative importance of each attribute to the overall quality (U) of the solution. This relationship can be expressed mathematically as follows:

$$CI_i = \frac{\partial U}{\partial A_i} \tag{7}$$

The importance ranking table display the relative importance that each engineering characteristics play in the overall quality of the design. It is of size 1 by *m*. The values displayed in this table are computed from the value stored in the relationship matrix and customer importance. Therefore the importance ranking defined as:

$$IR_j = \frac{\mathrm{D}U}{\mathrm{D}X_j} = \sum_{i=1}^{n} \frac{\partial U}{\partial A_i} \times \frac{\partial A_i}{\partial X_j} = \sum_{i=1}^{n} CI_i \times RM_{i,j} \tag{8}$$

The roof of the QFD is the correlation matrix allows the identification of trade-offs between engineering characteristics. It is build on the same format as the self interaction matrix. Therefore it is an upper triangular matrix of dimension m by m. Each cell specifies how one is going to change under the influence of another.

$$CM_{i,j} = \frac{\partial X_j}{\partial X_i} \tag{9}$$

Note: the correlation matrix is not taken into account in the importance ranking.

We can therefore conclude that the QFD facilitates the analysis of a concept by providing some form of a model relating lower level of the analysis (engineering characteristics) to the higher level (customer requirements and overall quality). By providing an overall ranking of engineering characteristics it provides guidance for further development.

Therefore the relationship matrix is based on the following assumptions:

- The relationships between attributes and engineering characteristics are monotonic and linear.

$$\frac{\partial A_i}{\partial X_j} = cst \qquad \textbf{(10)}$$

- There are no interactions between engineering characteristics in their effects on customer attributes
- The relationships between overall design quality and attributes are monotonic and linear.

$$\frac{\partial U}{\partial A_i} = cst \qquad \textbf{(11)}$$

- There are no interactions between attributes in their effects on overall quality.
- The propagated effect of one engineering characteristics (via its correlation with another engineering characteristic) is negligible.

In architecture conceptual design, the alternatives which will be considered are extremely different from one another. In each of these architectures the relationship between engineering characteristics are highly non linear. For example, if we consider the engineering characteristic “engine reliability” and the customer requirement “aircraft reliability”. Depending whether the aircraft has two, three or four engines, the relationship between the two will be different in amplitude. This observation is conflicting with first assumption above stating that interactions are monotonic and linear.

Also the QFD methodology does not work in situations were attributes influence the quality (or utility of the design) in a non-linear fashion. If we consider the subject of reliability, given the fact that reliability is subject to certification target, if the reliability is just below certification levels the influence the small change which will enable certification will influence the overall quality in a must stronger fashion than the same small change beyond certification point. This observation challenges the third assumption supporting the application of the QFD.

The nature of the assumptions on which a QFD is build makes its deployment to architectural trades hazardous. Moving from one architectural concept to the next implies moving from one extreme of the design space to the next. If we were to assume that there was an equation capturing the variation in attributes and quality, the QFD approximate this equation by using a local derivative. As we know from the Taylor series expansion rules, the further we go from the initial point, the least accurate a first order approximation is. Hence by the same token, the QFD methodology may provide accurate guidance for derivative design trades; but based on the reasoning presented above, I do not believe that it can support trade where different architecture concepts are considered.

3.3.1.2 Multi-Disciplinary Analysis

Multi-Disciplinary Analysis (MDA) is a type of model and a field of study. This field provides a nomenclature and toolbox for the organization and evaluation of large numerical models composed of multiple sub-elements. This field obtained its name from the fact that these large models are generally necessary to support multi-disciplinary trades. Unlike QFD, MDA is not a methodology. A Multi-Disciplinary Analysis is constituted of what are called “Collaborating Analyses” (CA). Each CA corresponds to a stand alone code which sends and receives input and outputs to other CA. The setup of the CA within the MDA is referred to as the Design Structure Matrix (DSM). Figure 55 represents an example of a DMS.

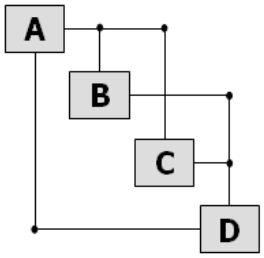


Figure 55: Example DSM

Inside this DSM we can see the four constituting CA represented as grey boxes. We can see that those boxes are connected by lines. Each line connects 2 boxes and represents the fact that the two boxes are exchanging information originating from the box with the line starting horizontally (either from the left or the right) and going to the box connected vertically (either its upper or lower side). In order to make sure that overlapping lines are not confused, each elbow is highlighted by a black dot.

In the example in Figure 55, the DSM represents an MDA composed of four CAs (referred to as A, B, C and D). Analysis A outputs information which is necessary to the evaluation of analysis B and C. The evaluation of A necessitates information produced by analysis D. Analysis D receives information from analysis B and C. A DSM can be formulated in different ways. Figure 55 presented an example, in what we can call its diagram form. The same information can also be presented in the table/matrix form presented in Figure 56.

	A	B	C	D
A	0	1	1	0
B	0	0	0	1
C	0	0	0	1
D	1	0	0	0

Figure 56: DSM in matrix form

It is important to note that the matrix form of the DSM matches the form of the interaction matrix which was presented earlier (in Figure 44). This parallel is very

important as it constitute a bridge between the conceptual definition and numerical analysis.

As it is often the case in MDA, the interconnections between CAs do not allow linear evaluation of the analysis. In the example in Figure 55, we can see that analysis A requires analysis D but at the same time analysis D requires analysis B and C which themselves rely on the estimation of analysis A. So in this example, there is a circular dependence which does not allow the linear estimation of MDA. This situation could have been directly spotted by the fact that there is a "feedback connection". In this example, the feedback connection is the connection from D to A. If we assume that analyses are triggered along the diagonal from top left to bottom right, a feedback connection refers to the fact that a later CA must provide information to an earlier CA. Other connections are considered as "feed forward".

3.3.1.2.1 Convergence of MDAs

In order to evaluate an MDA including feedback connections, iterative evaluation is necessary. Several methods enable convergence to a solution. The following section will describe some of them and will compare their advantages.

3.3.1.2.1.1 Fixed point iteration:

This implies that if some piece of information is supposed to be fed back, a "guessed" value is fed forward during the first iteration. After all analyses were evaluated, the values calculated at the previous iteration are used for feedback values. The iterative process then proceeds until a stability condition on feedback values is reached (the newly calculated estimation is within an error margin from the previous estimation).

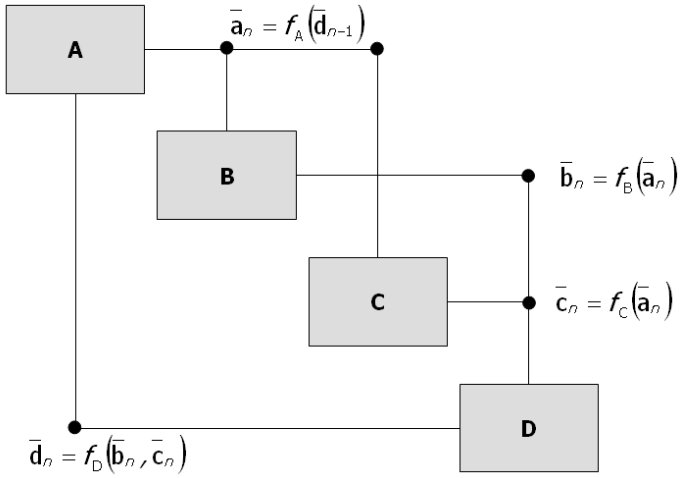
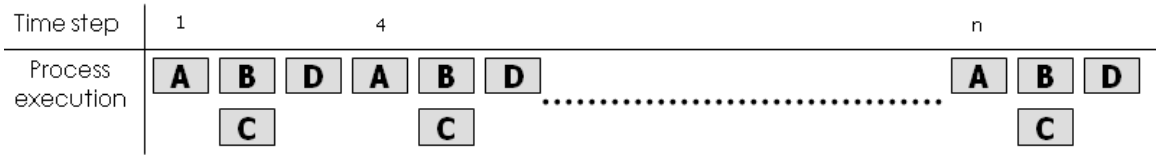


Figure 57: Fixed point iteration for MDA convergence

In order to solve the MDA presented in Figure 57, the following sequence would be necessary to converge on a solution. Note some aspects of the MDA can be performed in parallel. In the example in Figure 57, analyses B and C can be performed in parallel.



3.3.1.2.1.2 Optimizer based decomposition:

The optimizer decouples the CAs, and feeds simultaneously input values determined concurrently by an optimizer. The optimizer operates on a compatibility constraints defined by the vanishing Jacobians (squared difference between target and calculated value of linking variables). The resulting modified MDA is represented in Figure 58.

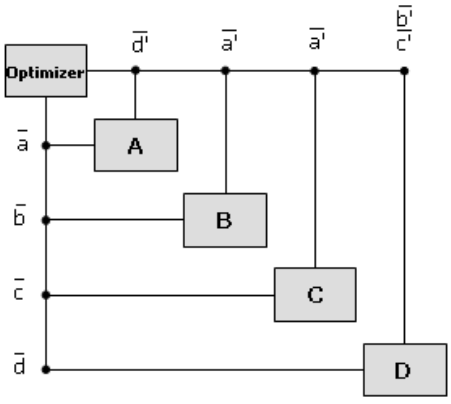
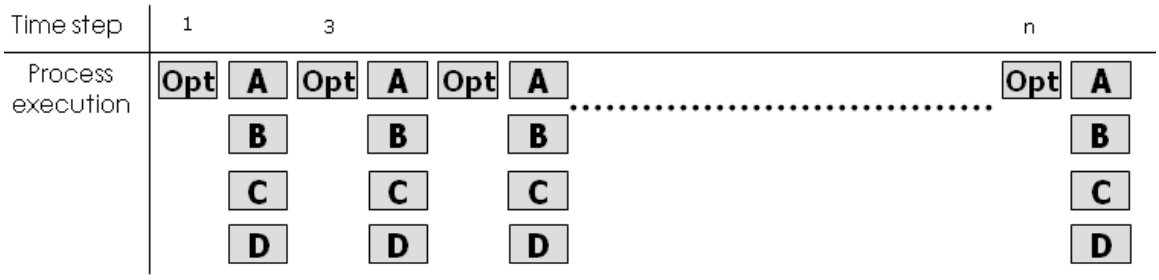


Figure 58: Optimizer based decomposition

The optimization problem for the convergence optimizer is the following:

$$\underset{\bar{a}',\bar{b}',\bar{c}',\bar{d}'}{\underline{Min}} \quad \left(\bar{a}-\bar{a}'\right)^2+\left(\bar{b}-\bar{b}'\right)^2+\left(\bar{c}-\bar{c}'\right)^2+\left(\bar{d}-\bar{d}'\right)^2 \tag{12}$$

The advantage associated with this approach is the opportunity to parallelize the execution of the CAs.



3.3.1.2.2 Multi-Disciplinary Optimization

The optimization of an MDA is a delicate matter. Given the fact that an MDA may require a first iterative process to converge on each set of global design variables, the optimization of the MDA requires adding a new layer of iterative investigations. If we consider these two iterative processes, the total number of iteration necessary (N_{opt}) is equal to the product of the number of iteration necessary to converge (n_c) by the number of iteration necessary to the identification of an optimum (n_{opt}).

$$O(N_{opt}) = O(n_c) \text{ x } O(n_{opt}) \tag{13}$$

This multiplicative increase highlights the computational complexity associated with the optimization of an MDA. In order to address this situation several optimization methods were proposed by the Multi-Disciplinary Optimization (MDO) community.

In order to compare the optimization strategies, let us consider how they address the following MDO problem:

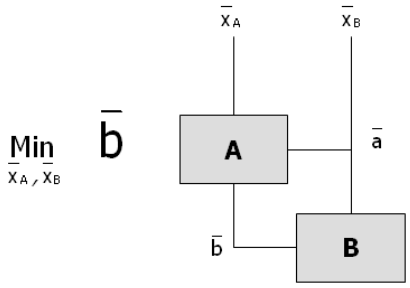


Figure 59: Sample MDO

3.3.1.2.3 All-in-One Optimization Strategies

The "all-in-one" (AiO) optimization process is structured around one optimizer. In this process, the MDA is considered as a black box where design variables X are fed-in and which returns the objective value.

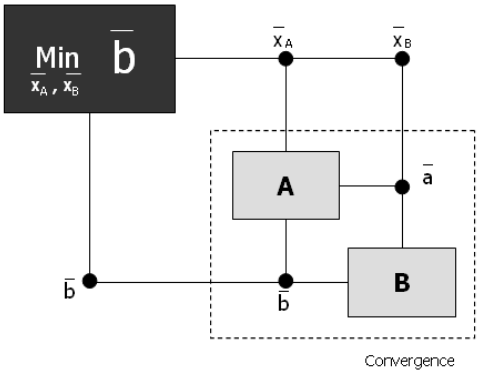


Figure 60: All in one optimization setup

3.3.1.2.4 Multi-Level Optimization Methods

The multi-level optimization methods consist in placing an optimizer within each CA. An optimizer at system level uses the linking variables as design parameters. The vanishing Jacobians are used as constraints and integrated to the system level objectives as penalties. The responsibility of identifying the optimal set of design variables is delegated to the CA-level optimizers. Their objective is to match the value of the linking variables values provided by the system level optimizer (minimization of their vanishing Jacobian) [50].

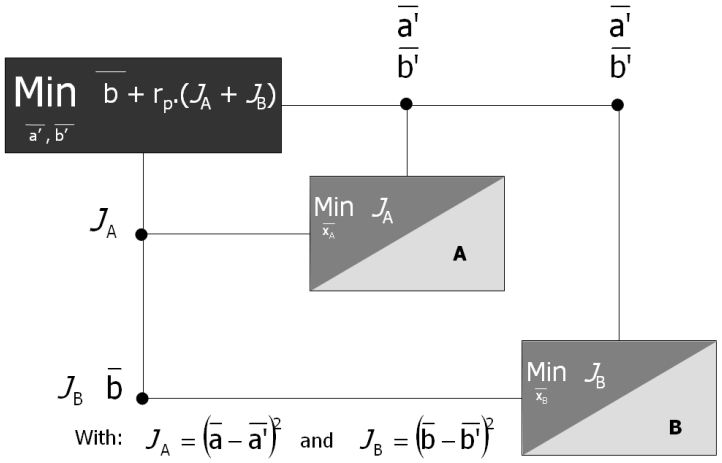


Figure 61: Collaborative optimization setup

Note on Figure 61: The system level optimization problem will also include constraints on the value of the vanishing Jacobian.

The system-level optimizer sets targets on linking variables and the CA-level optimizers try to match these targets. The CA-level optimizers influence the system-level optimizer by returning the value of their Jacobian. If the Jacobian value is too high it will increase the objective function of the system-level optimizer. This increase shall suggest to the system-level optimizer that the target for the linking variables was unrealistic. This method addresses simultaneously the optimization of an objective function and the convergence of an MDA with feed-back.

An example of such a method is the Analytical Target Cascading. This method is based on a multilevel optimization scheme developed at university of Michigan [51], [52], [53] and [54]. This methods is of particular interest to this work because its goal is "to propagate desirable overall product targets to appropriate individual specifications for the various subsystems and components" [51]. This goal is in line with *Objective 5* (to reduce development risk by formulating validated guidelines driving detailed developments).

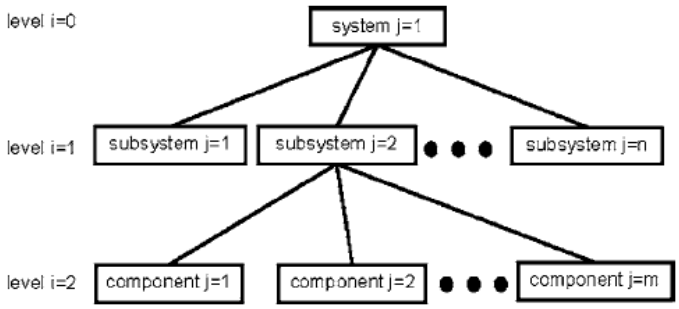


Figure 62: ATC hierarchy

The ATC method is based on multiple levels of optimization. Each level has one parent and possible multiple children. The design variables in the optimization problem of the parent include its own design variables plus target for the development of its children. On the children side the target defined by the parent will drive their definition. Since the definition of the children influence the performance of the parent. Therefore, if the targets are appropriately defined by the parent, the children optimization process is going to contribute to the optimization of the parent. At the end of the optimization process, the designer has identified design targets for each component which will contribute to the optimization of the system as a whole.

There are many forms of multi-level optimization strategies. Each of these strategies formulates the system-level and CA-level optimization problems differently. Some methods like BLISS (Bi-Level Integrated System Synthesis) [55-56] use the local sensitivities of the objective function with respect to the various linking variables. By providing these sensitivities the CA-level optimizer can be granted more flexibility in their optimization process which accelerates the convergence process.

3.3.1.2.5 Sequential Optimization

The sequential optimization is more a mathematical illustration of serial development bad habits rather than an optimization strategy per se. The sequential optimization approach optimizes each CA at a time. By doing so this optimization strategy attempts to emulate a serial development (by opposition to a concurrent

development). In [57], Sobievsky and Kroo compared results obtained from AiO, CO and sequential optimization strategies which presented the fact that sequential optimization will have difficulty reaching an optimum. It is interesting to observe that this comparison provides some of mathematical proof to the statements made in Chapter 2.

3.3.1.2.6 Discussion

Reference [58] provides a comparison of the performance of the various optimization strategies. The observations by de Wit et al. show that, from a computational standpoint, the All-in-One approach is more efficient at finding an optimum than any of the multi-level optimization strategies. This is explained by the fact that each iteration of the system-level optimization process requires solving a set of optimization processes at CA level.

I believe that general rules about the efficiency of one optimization method to another can not be made. “All in one” approaches have been proven to be the most robust optimization strategy. But in situation where the convergence is difficult to reach or where transfer of information between CAs is expensive, the collaborative optimization strategies offer appropriate solutions to optimize during the convergence process and in the mean time to reduce the amount of necessary exchange between CAs.

3.3.1.3 Discussion on Modeling Approaches

In this section we have observed different approaches to estimate the performance and attributes of architecture concepts. Given the scale and technical complexity of an aircraft system architecture is it difficult to apply a QFD approach. Although the QFD is a means to synthesize information at architecture level it still assumes that the emergent responses at architecture-level can be captured by the experts. In fact, experts understand the lower level of the problem based on their knowledge about subsystem design. Relating their level of knowledge to the level of synthesis necessary to the qualification

and comparison of architecture alternatives requires more flexibility than the linear assumptions used by a QFD.

On the other hand, Multi-Disciplinary Analysis techniques and methods can integrate this information. The expert knowledge at the subsystem level can provide the Collaborating Analyses which can then be associated to provide a complete representation of the architecture. Hence the subsystem level can be synthesized to capture the emergent information at architecture level. Also the MDA techniques have been implemented in commercially available software. Among them we find environments like iSight (from Dassault Systèmes [59]), Model Center (from Phoenix Integration [60]), or PaceLab Suite (from PACE [61]).

It can be concluded that the MDA methods and techniques provide a rigorous framework which can be used to build the analysis of aircraft system power architectures. The modular approach to modeling it implies is an important asset in a context where the knowledge is scattered among different individuals. Its analytical generality also has the potential to support the large technical complexity implied by the scale of aircraft power system architectures. The maturity of this field also provides commercially available and industry tested solutions. Based on this conclusion on the analysis framework to be used in this dissertation, we can now observe the different approaches to the analysis of system architectures.

3.3.2 **Analysis Approaches**

When we analyze large scale architectures, two approaches can be considered. When one considers a new concept, the attributes and performance of this concept can either be derived by the analysis of its components up (bottom-up analysis) or by drawing conclusions from a design that is sufficiently similar to the one of interest (derivative analysis).

3.3.2.1 Bottom-up Analysis

In engineering developments, we generally consider the architecture as the "top-level" aspect of the design process and the subsystems as "lower-level". The bottom-up approach consists in integrating subsystem-level knowledge in order to capture the emergent attribute of the architecture. This is performed by sizing subsystems in a first step, then synthesizing their attributes in order to define the aircraft level attributes. This approach is modular and allows the decomposition of the analysis into subsystem sizing processes which tend to be more accurate that their aggregated sizing implied by the top-down approach (sizing of the subsystems based on the characteristics of the aircraft). Using a bottom-up approach enables the use of a modular approach which can be adjusted to the architecture concept under consideration. This modularity is an important asset when we want to consider different architectures.

3.3.2.2 Derivative Analysis

Derivative analysis is based on the consideration of a baseline on which modifications are applied at subsystem level in order to meet new requirements [62-63]. The analysis is based on the leading thread defined by the **Propagation of Change** (PoC). If a subsystem element requires a modification, the PoC consists in identifying the changes within the architecture necessary to accommodate or adjust to this modification.

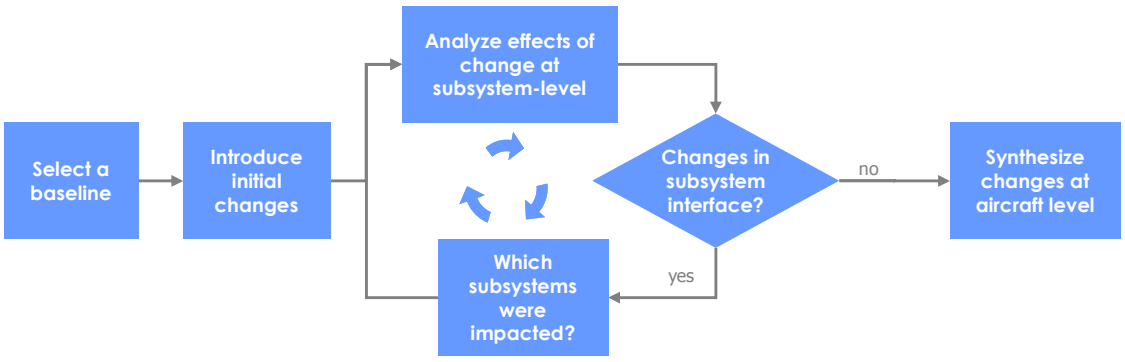


Figure 63: derivative analysis process

The fact that the analysis is initiated with a known baseline, provide an initial framework to integrate the subsystem investigation. Therefore as changes are introduced, the analysis efforts may be focused on specific subsystems while other elements are assumed to be unchanged. As the initial changes are implemented, collateral adjustments are necessary. As collateral changes are implemented the changes propagate within the architecture. Once all necessary changes have been considered at subsystem level, they are synthesized to correct the baseline attributes at aircraft level.

In the field of engineering changes, the propagations of a modification within the architecture are classified under three categories [64]: the avalanche, the blossom and the ripple changes

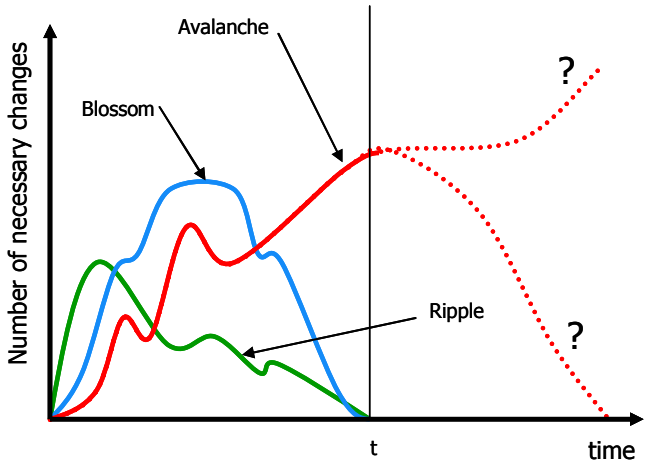


Figure 64: Types of change propagation

The figure above represents the changes during the development process which is a different context that the one of a specific analysis. In a development the changes will be captured as it progresses. In the context of an architecture analysis the full propagation is expected to be captured. Therefore the propagation of the type “avalanche” which is difficult to contain in a development process does not agree with the assumption that the propagation of change should be bounded.

It is important to consider that the derivative approach assumes that the propagation of change is bounded (i.e. the iterative process in Figure 63 has an end). In

order to do so, the baseline architecture must be sufficiently similar to the final architecture to justify applying correction factors on its attributes. It also assumes that the set of "Necessary" changes are identical to "optimal" changes. In a situation where a technology, such as more electric actuators, is introduced in a traditional ASA, the changes necessary to optimize the architecture will deeply modify it. The hydraulic power architecture will disappear; the electric power architecture will be fundamentally modified. Therefore when the technologies considered have such profound impact on the architecture, the assumption that capturing the propagation of change is probably inappropriate.

3.3.2.3 Review of Analysis Approaches

Based on was observed earlier, the bottom-up approach tends to be used for the analysis of smaller systems where information at the subsystem-level is available. On the other hand the derivative approach is used for the analysis of large problems where a baseline is available. The design problem considered in this thesis falls partly under the typical use of both approaches. The aircraft power system architecture is a large design problem where baselines are available (previous aircraft programs). But in a context of architectural design, it is difficult to relate the new concepts to the old baseline. This difficulty implies long propagations of changes which are difficult to keep track of. This observation constitutes a main showstopper for the use of derivative approaches.

At the same time bottom up approaches are generally used for simple systems where the analysis and synthesis of their subsystems is possible. This method tends to be applied mostly to simple (or simplification of large) systems. This tendency can be explained by the integration challenge it implies. In order to perform a bottom-up analysis it is necessary to properly integrate the lower level parts of the analysis. The larger the system is, the more knowledge to integrate. The system considered in this thesis (aircraft power systems) is large and technically complex. However, it is assumed

that the subsystem knowledge can be provided by experts constituting the IPT performing the trade (see chapter 2).

This thesis will use a bottom-up approach. This choice is made possible by the assumed availability of subsystem models via the participation of experts constituting the IPT. But at this point, the practicality of such an approach remains an open question due to the scale of the design problem considered in this work. Therefore, the integration of subsystem knowledge and setup of models analyzing different architectures is a central question that will need to be addressed in this thesis. This question is captured directly by *Research Question 2*.

3.4 Review and Example of Conceptual Architecting Processes

In this chapter, we have analyzed different methods, techniques and tools which can be used in architecture developments. To provide the reader with a rapid overview of the state of the art, a matrix of alternative was prepared. This matrix lists by row the different methods available to support the various activities implied by conceptual design.

Table 12: Methods, techniques and tools alternatives

		ALTERNATIVES				
Requirement formulation	Organization	Functions	Objectives	Functions & Objectives		
	Formulation	Targets	Objectives	Objectives & Targets		
Architecture concept definition	Breakdown	Disciplinary	Physical	Perimeter-based	Functional	ATA chapter
	Composition	Basic matrix of alterntives	IRMA	ARM	BDD (SysML)	BDD + MMM
	Structure	Implicit (ignored)	Structure matrix	Genetic network strings	IBD (SysML)	IBD + MMM
Assessment	Optimization method	stochastic method	Deterministic methods	Design exploration		
	Analysis approach	Derivative	Infusion method	Bottom up	No analysis	

In order to complete this overview of the state of the art, this section will review some methodologies proposed in the field of conceptual design of ASA. In order to facilitate the description of what they contain, the composition of each methodology is presented as a selection in this matrix of alternatives.

3.4.1 **Technology Mapping Methodology Roadmap**

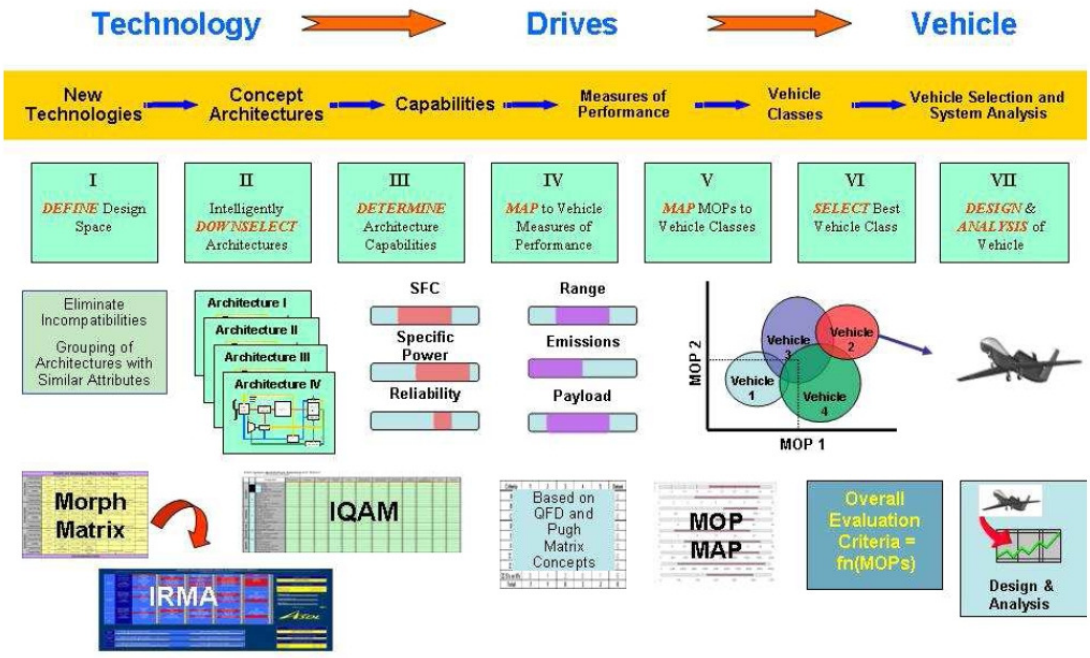


Figure 65: Technology Mapping Methodology Roadmap

This method was proposed by Soban et al. and is presented in more details in reference [65] and [66]. It was applied to the development of fuel cell based power architecture in Uninhabited Aerial Vehicles (UAV). This method is based on a functional decomposition of the architecture which allows defining an IRMA. The IRMA facilitates the elimination of obviously suboptimal alternatives and allows the designer to focus on "intelligently" defined architectures. The technology portfolio from which the IRMA composes the architecture is defined using a relational matrix. The relational matrix determines the "architecture capabilities" which correspond to vehicle attributes (Specific Fuel Consumption, Startup time, reliability). These vehicle attributes are then fed in a mission analysis tool which translates these attributes into Measures of performance. Using an OEC, the best architecture concepts were selected and passed on to further analysis. An overview of the techniques and tools constituting this methodology is presented in the following matrix.

		ALTERNATIVES				
Requirement formulation	Organization	Functions	Objectives	Functions & Objectives		
	Formulation	Targets	Objectives	Objectives & Targets		
Architecture concept definition	Breakdown	Disciplinary	Physical	Perimeter-based	Functional	ATA chapter
	Composition	Basic matrix of alterntives	IRMA	ARM	BDD (SysML)	BDD + MMM
	Structure	Implicit (ignored)	Structure matrix	Genetic network strings	IBD (SysML)	IBD + MMM
Assessment	Optimization method	stochastic method	Deterministic methods	Design exploration		
	Analysis approach	Derivative	Infusion method	Bottom up	No analysis	

Figure 66: MA choices for Technology Mapping Methodology

This methodology allows the designer to explore architectures where limited technical information is available and provides a traceable way to the selection of the most promising architectures. In this example, the lack of redundancies in the subsystems composing the considered architecture allows the designer to abstract away the structure of the architecture.

If this method, allows the identification of promising technology mixes, it does not clearly guide the development of architecture developments. The values required to constitute the relational matrix can be difficult to provide as the mathematical model formulated it is built upon does not capture correlated effects between technologies (i.e. the effect of technology A in association with technology B is not necessarily the same as the effect of technology A in association with C).

3.4.2 **Quantified Relationship Matrices for Fuel Systems**

Gavel et al. proposed a methodology for aircraft fuel systems design in the following publications [67], [68] and [69]. The methods is based on a tool implemented in an excel spreadsheet which allows the definition of the architecture and provides the designer with attributes qualifying the performance of the resulting architecture.

In order to implement architectural trade-offs, a matrix of alternative was used to formulate the composition of the architecture. The structure of the architecture was defined using a QFD matrix was adapted with an interaction matrix. This implementation allowed the automated formulation of the I/O structure necessary to perform a numerical analysis of the architecture.

	1	2	3	4	5	Choose
Engine feed	NGT	HOPPER-tank	HT with Jet pumps	NGA		1
Fuselage Tank Transfer	Distributed pump	Inline pump	Jet pump	Gravity	Siphoning	1
Wing Tank Transfer	Distributed pump	Inline pump	Jet pump	Gravity	Siphoning	1
Drop Tank Transfer	Distributed pump	Inline pump	Jet pump	Gravity	Siphoning	1
Σ Transfer						
Vent & Pressurization	Closed system	Ejector system	Non Pressurized			1
Measurement	Level sensor	Tank probe	Both			3
Refueling	Pressurized	Gravity	AAR			1
Fire P. Fuselage & Wing	SAFOM	OBIGGS	Liquid Nitrogen	None		4
Fire P. Drop Tank	SAFOM	OBIGGS	Liquid Nitrogen	None		4

Figure 67: Matrix of alternative for Fuel Systems [68]

Pump:	A	B	C	D	E
A Pressurization	A				
B Engine feed	x	B			
C Vent System			C	x	
D Refueling			x	D	
E Fuel transfer	x				E

Figure 68: Interaction matrix for fuel systems [67]

The work by Gavel et al. offer the proof that matrix of alternatives in association with interaction matrices can both describe an architecture concept and define the I/O model necessary to the formulation of a numerical model.

The combination of tools used in the solution proposed by Gavel et al. is the following:

		ALTERNATIVES				
Requirement formulation	Organization	Functions	Objectives	Functions & Objectives		
	Formulation	Targets	Objectives	Objectives & Targets		
Architecture concept definition	Breakdown	Disciplinary	Physical	Perimeter-based	Functional	ATA chapter
	Composition	Basic matrix of alterntives	IRMA	ARM	BDD (SysML)	BDD + MMM
	Structure	Implicit (ignored)	Structure matrix	Genetic network strings	IBD (SysML)	IBD + MMM
Assessment	Optimization method	stochastic method	Deterministic methods	Design exploration		
	Analysis approach	Derivative	Infusion method	Bottom up	No analysis	

Figure 69: MA choices for Quantified relationship matrices

3.4.3 **Strategic Technology Planning**

The strategic Technology Planning process (STeP) was proposed by Kirby et al.[70]. It allows for the translation of vehicle-level (i.e. system-level) objectives into technological choices. An overview of the step implied by this process is presented in the following figure:

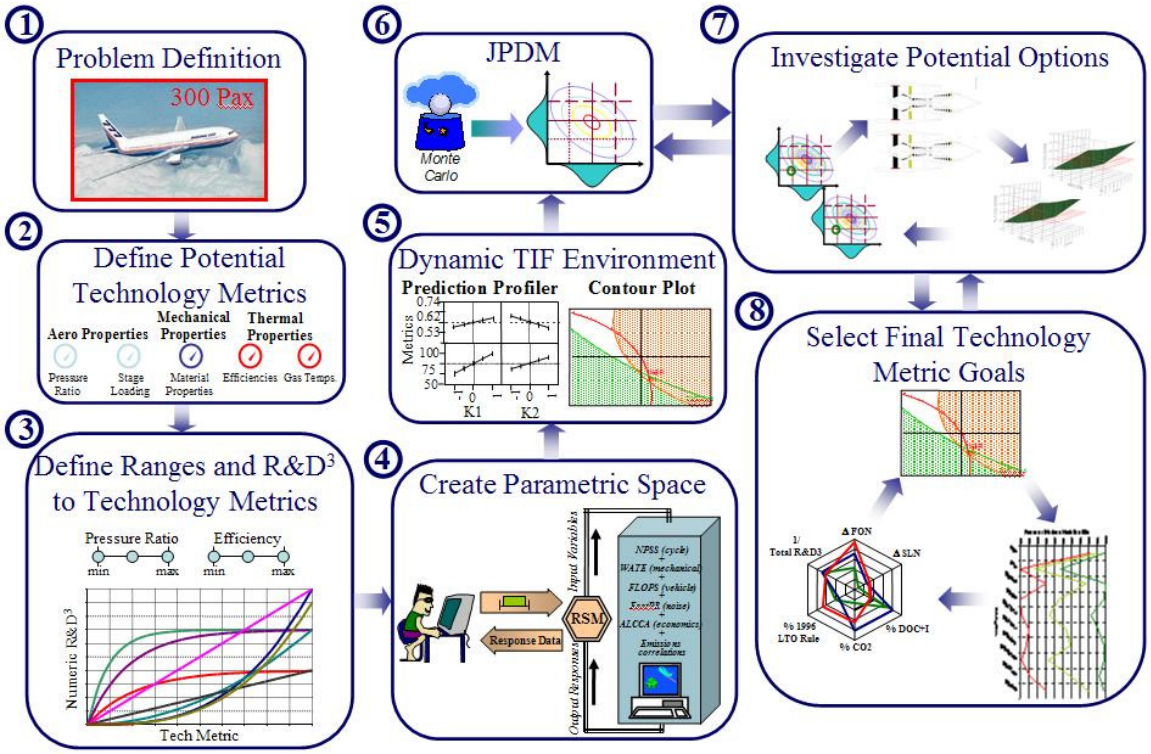


Figure 70: STeP process

The great advantage of the STeP process was the fact that it clearly recognized that two levels of trades exist (the aircraft or system-level and the technology or subsystem-level) and that the translation of an objective at one level does not easily translate into the other. The main outcome of this process is not a design concept in the traditional sense (i.e. the aspect ratio of the wing must be X, the bypass ratio Y, etc…) but a technological strategy in the form of % improvement for future developments. Therefore instead of producing a design frozen in geometrical parameters (which

inevitably need to be refined in later phases), it focus on how the development must be performed.

		ALTERNATIVES				
Requirement formulation	Organization	Functions	Objectives	Functions & Objectives		
	Formulation	Targets	Objectives	Objectives & Targets		
Architecture concept definition	Breakdown	Disciplinary	Physical	Perimeter-based	Functional	ATA chapter
	Composition	Basic matrix of alterntives	IRMA	ARM	BDD (SysML)	BDD + MMM
	Structure	Implicit (ignored)	Structure matrix	Genetic network strings	IBD (SysML)	IBD + MMM
Assessment	Optimization method	stochastic method	Deterministic methods	Design exploration		
	Analysis approach	Derivative	Infusion method	Bottom up	No analysis	

Figure 71:MA choices for STeP

However this approach is limited by its weak modeling approach which is based on a simple parametric analysis. In this approach, the impact of the technology is infused parametrically on a fixed model which assumes a fixed architecture.

3.4.4 **A Simulation Framework for Aircraft Power Management**

This work was performed by Susan Liscouet-Hanke [22, 71-72] as part of her thesis work in the future project group at Airbus S.A.S.

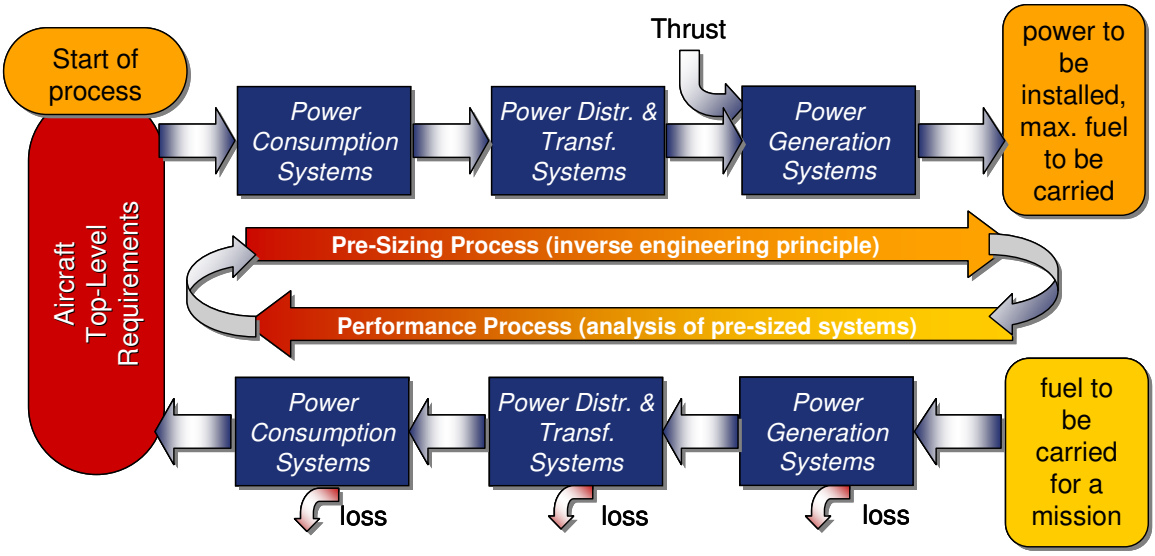


Figure 72: Integrated sizing and performance process

This approach was based on a functional breakdown of the architecture. The architecture was defined based on a physical description of functional groups. These groups were represented by modeling elements integrated in Simulink in order to perform the analysis process represented above in Figure 72. These modeling elements were integrated automatically based on the physical alternatives chosen to implement the functionality.

The main limitations associated with this approach results from the functional approach used in the composition of the model supporting the analysis. Previously, we observed that the functional decomposition of the architecture provides an instable breakdown of the physical elements composing it. Depending on the physical implementation on some functions other functional groups may or may not be necessary. In order to address this problem, Liscouet-Hanke had to use large subsystem groups. This

large decomposition made the subsystem models difficult to define and recycle since they were often architecture specific.

Nevertheless, Liscouet-Hanke's thesis has been an important breakthrough in the field of architecting as she has been the first to demonstrate the effectiveness of numerical approaches in supporting architectural trade-offs. Hence, her work has, in many ways, been an inspiration of the ideas presented in this thesis. An overview of the methods used by Liscouet-Hanke is provided by the following matrix of alternatives.

		ALTERNATIVES				
Requirement formulation	Organization	Functions	Objectives	Functions & Objectives		
	Formulation	Targets	Objectives	Objectives & Targets		
Architecture concept definition	Breakdown	Disciplinary	Physical	Perimeter-based	Functional	ATA chapter
	Composition	Basic matrix of alterntives	IRMA	ARM	BDD (SysML)	BDD + MMM
	Structure	Implicit (ignored)	Structure matrix	Genetic network strings	IBD (SysML)	IBD + MMM
Assessment	Optimization method	stochastic method	Deterministic methods	Design exploration		
	Analysis approach	Derivative	Infusion method	Bottom up	No analysis	

Figure 73: MA choices for a simulation framework for aircraft power management

3.5 Conclusions

The state of the art review provided in this chapter has shown that some methods have the potential to be key enablers for architectural trades. The requirement definition activity requires both the use of functional analysis to constraint the design analysis and of objectives to guide the design process in selecting architecture and subsystem alternatives.

We have also observed that the functional decomposition organizes the mission requirements and initiates the architecture definition. This functional decomposition must be accompanied by a functional analysis of the subsystem physical solutions which can then be used as the building block for the architecture concept definition. These building blocks can be assembled using SysML to describe architecture concepts in a graphical and unambiguous fashion.

The architecture concept description can be used for the definition of a model. This model can be defined based on expert analysis using MDA methods. These methods allow for the integration of knowledge necessary to bottom-up developments. Using bottom-up developments rather than a derivative design approach does not necessitate similarity between the concept considered and an arbitrary baseline. This important aspect directly supports the fundamental research *objective 1* and *4*.

The integration of the methods indicated above implies addressing several important technical gaps. A review of these gaps introduces Chapter 4. This chapter will be dedicated to the description of the methodology proposed in this thesis.

Chapter 4

Proposed Methodology

The purpose of this chapter is to introduce the methodology proposed in this thesis. This chapter starts by reviewing the general framework considered for architecture conceptual design and the challenges associated it. Based on this framework description, a methodological process will be formulated to address these challenges. The first section will be dedicated to the overview of the framework and introduction of the methodology. The following sections will each focus on a different aspect of the process.

4.1 Introduction

In chapter 2, the following assertion was proposed:

The objectives can be achieved using an IPT to conduct conceptual design trades, based on an analysis performed by an MDA composed of numerical models defined and controlled by the experts composing the team.

In order to define the context in which the methodology proposed is implemented, the IPT/MDA framework implied by the assertion above is described in this section. From this description, the complete set of research questions will be reviewed. This review will open the way for the introduction of the methodology proposed in this thesis.

4.1.1 **Proposed Architecting Framework**

The composition of the framework proposed in assertion 1 is shown in Figure 74. This framework is based on an IPT and an MDA. The IPT is the **Architecting Team** (AT) as described in Chapter 2. The AT is composed of two types of actors: The expert and the architect.

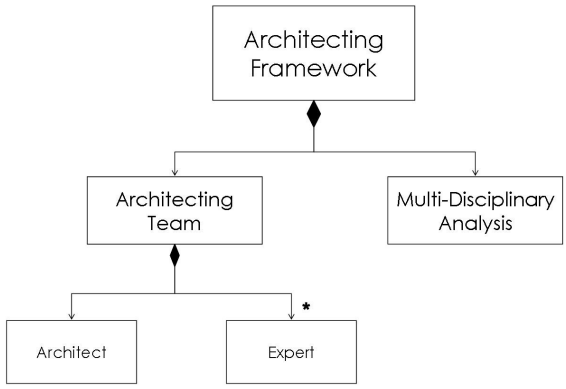


Figure 74: Architecting framework composition

This framework is based on four roles:

- The AT (collective role for experts and architect)
- The architect (or aircraft architect)
- The expert (or system architect)
- The MDA

The AT's role refers to the coordinated actions of the experts and the architect. Together their role is to define architecture concepts based on the analysis of previous architectures. The role of the AT is represented in a graphical form in Figure 75. From this point, the AT will be represented as a symbol with an oval around the letter "AT" (see below).

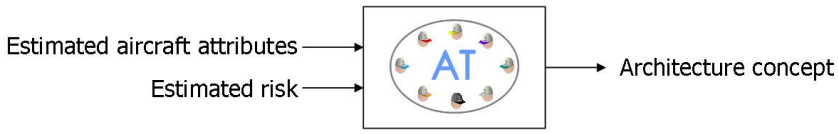


Figure 75: Architecting Team role

The **architect**, in addition to his/her role within the AT, is responsible for specifying and enforcing aircraft requirements. These requirements are susceptible to be reviewed based on feasibility information from the analysis. From here on, the symbol representation of the architect will be a head with a black hat (see Figure 76).

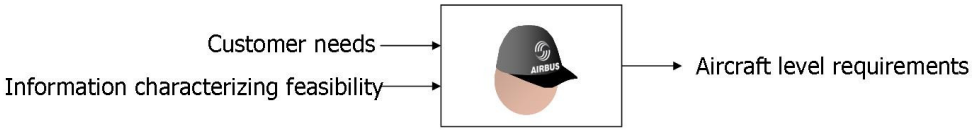


Figure 76: Architect role

The **expert** provides subsystem sizing models and contributes to the AT. The sizing models should be continuously adjusted during the study as the context of integration of the subsystem and the evaluation of risk evolves. The symbol for the experts will be a head with a colored-visor hat.

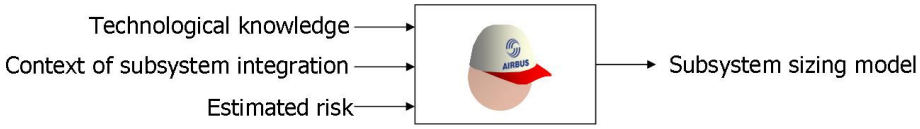


Figure 77: Expert role

The **MDA** is the processor of the information delivered by the actors presented earlier. The MDA receives as input the requirements formulated by the architect, the architecture concept formulated by the AT and the subsystem sizing models provided by the experts. The role of the MDA is to compile and synthesise the information necessary to the decision making process of the AT, architect and experts.

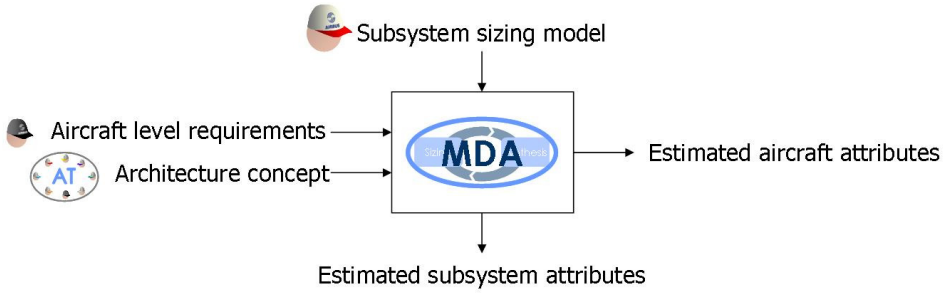


Figure 78: MDA role

4.1.2 State of the Art Gaps in the Deployment of this Approach

In previous pages, the main roles of the proposed framework were introduced. Before we move on to a more detailed description of the proposed methodology, we shall introduce the main gaps and challenges associated with the implementation of the

proposed framework. From these gaps, research questions will be derived. These research questions will then lead us to the methods proposed in this thesis.

4.1.2.1 Formulating the Mission

In order to properly analyze the architecture, it is necessary to accurately size subsystems. The requirements imposed upon their definition will influence their sizing and consequently the accuracy of the estimation of architecture attributes. Therefore, it is essential to consider all forms of operations that can be expected by the mission. When we consider a system as complex as an aircraft, inaccurate conclusions will be reached if the mission is reduced to normal operation. Often subsystems are not sized for their standard operating scenario, but rather designed to handle failure states in non-standard conditions (with a failed subsystem, hot/cold weather operations, etc…). It is therefore necessary to capture and characterize the ensemble of scenarios in which the aircraft is expected to operate. The following research question must therefore be addressed:

Research Question 1: How can we characterize the mission in a fashion that captures all requirements driving subsystems sizing?

4.1.2.2 Automated Setup of the Model

Objective 4 implies that multiple architecture concepts must be considered (in order to detect architectural opportunities). The consideration of different architecture concepts implies different compositions and different relationships between subsystems. As different compositions and relationships are considered, the requirements and rules on which subsystems are sized change. Therefore, it is impossible (impractical) to have a single, stand-alone model capturing the sizing of multiple architecture concepts. This observation allows me to conclude that it is necessary to generate an MDA for each architecture concept considered. The necessity to define a new MDA for each architecture concept leads us to the following research question:

Research Question 2: How should we define the architecture alternatives in an unambiguous fashion while facilitating the setup of the model?

4.1.2.3 Encapsulating Subsystem Knowledge

The modeling approach proposed in this work assumes that the architecture analysis can be constituted automatically based on modeling bricks. These "bricks" are subsystem sizing models. Therefore, this overall approach requires that some degree of expert knowledge, associated with subsystem design, can be encapsulated in a numerical model. In order to do so, it is necessary to clearly understand and formalize the way subsystems are sized. Using a mathematical formalism provides an unambiguous process which can be then deployed on all forms of expertise necessary to the architecture analysis. This observation leads us to the following research question:

Research Question 3: How can we mathematically formalize subsystem sizing while allowing for their coordinated optimization?

4.1.2.4 Setting up Subsystem Design Problem

If one of the main objectives of the work is to optimize the architecture, the other is to integrate the conceptual design trades in the industrial development of the architecture. In chapter 2 we have observed that the subsequent development of the architecture is performed in a clustered concurrent engineering approach. In order to successfully deploy this approach, the conceptual design phases must formulate the design problem associated with each subsystem. This observation leads us toward the following research question:

Research Question 5: How do we translate architecture level objectives into specifications and guidelines to the subsystem developments?

4.1.3 **Overview of Proposed Process**

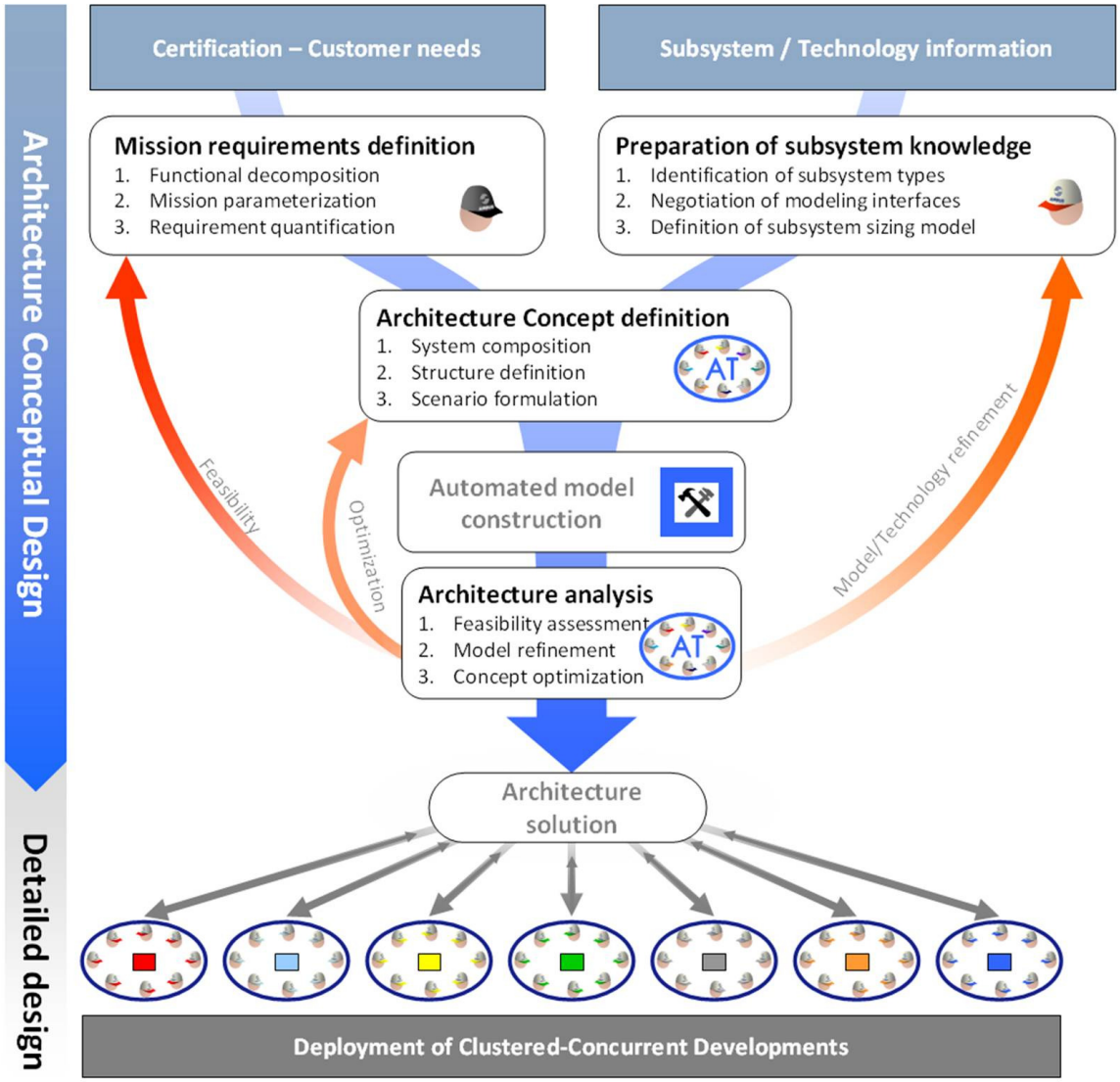


Figure 79: Organization of the overall process

The process is initiated by the architect with the mission requirement definition. This phase and its activities are described in section 4.2 of this chapter. They consist in decomposing the mission into functions (boundary functions) and organizing the associated requirements (functional requirements).

In parallel with this activity, the experts initiate the integration of subsystem knowledge. This activity is described in section 4.3. The experts define a list of

subsystems that may be considered for architecture concept alternatives. Each of these subsystems is assigned to system experts. The system experts, with the support of their development team, setup models representing the design of their subsystems (sizing models). As these models are defined the subsystem experts meet regularly to establish standard modeling interfaces. This activity is concluded through the establishment of a model library which can be used for the architecture analysis.

Using this library, the architecting team can define architecture concepts. This definition process is described in section 4.5. In this activity, the architecture concept is defined graphically using the System Modeling Language (SysML). As the architecture is defined, a builder will access the library defined previously. Based on methods proposed in this thesis the builder will automatically construct a model (MDA) corresponding to the concept described. The methods supporting the builder logic are presented in section 4.6.

Executing the MDA will size and analyze the architecture concept described by the architecting team. Based on its results, the architecting team obtains an estimation of the attributes of the architecture. This estimation allows for the comparison and analysis of architecture concept alternatives. This architecture analysis guides the architecting team in optimizing the concept. In addition to architecture level information, each expert can access their subsystem models. They can observe which constraints and operational scenarios have driven their design and which attributes were identified as critical. Using simultaneously the information at aircraft/architecture-level and subsystem-level, the architecting team has the information necessary to improve their architecture concepts. This architecture analysis will be described in section 4.7.

Once an architecture concept is chosen its corresponding MDA provides subsystem design requirements for subsystem developments. Together the architecture concept and guidelines define an architecture solution which defines and guide the Clustered Concurrent Engineering development described in chapter 2. Then, in addition

to the identification of the most suitable architecture concept, the outcomes of this process are:

- The definition of the functional specifications defining the mission of subsystems
- An initial design (or baseline) for subsystems
- An objective function which can be used to guide the expert in subsystem-level trade-offs

4.2 Architecture Requirement Definition

The requirements driving the sizing of subsystems are often difficult to identify. Most subsystem sizing constraints will not be driven by normal operation but rather by unusual (and sometimes unexpected) combinations of failures and operating conditions. Previously, this observation led us to the following research question:

Research Question 1: How can we characterize the mission in a fashion that captures all requirements driving subsystems sizing?

To address this research question, the challenges imposed by the complexity of the mission will be broken down into three aspects. The first aspect is the diversity in type of needs implied by the mission (functional decomposition). The second aspect captures variation of the operations in the mission (mission parameterization) and the third qualifies the amplitude of requirement for the scenarios composing the mission (requirement quantification).

Figure 80: Process overview - Mission requirement definition

4.2.1 **Identification of Boundary Functions and Objectives**

In Chapter 3, we observed that there are two types of requirements:

- Negotiable requirements
- Non negotiable requirements

Together these two types of requirements define a mission. Ignoring one form of requirements may result in an ill-defined formulation of the mission. Based on this observation the methodology proposed will make sure that it captures both forms of requirements.

For each an appropriate tool was identified. Negotiable requirements are properly captured by figures of merit and non-negotiable requirements by functional specifications. Therefore every time a mission is defined (at aircraft/architecture-level or subsystem level) this format will be used.

It was also observed that when a requirement must be met in order to validate the integration of an element into a larger architecture, the requirement can be considered as non-negotiable. Otherwise, all other requirements are assumed to be negotiable. The consequence of this observation is that for subsystems the requirements related to their integration must be formulated as functional specifications (non-negotiable).

4.2.1.1 Functional Decomposition

If we observe the mission of an aircraft, we can see that it consists of an ensemble of tasks that must be realized in order to have a valid architecture. For instance, the mission of a commercial aircraft implies that thrust must be provided and that cabin is pressurized. The identification of these tasks is the product of the functional analysis of the mission. This analysis allows for the breakdown of the mission into smaller and simpler “sub-mission” elements called **functions**. This breakdown greatly simplifies the consideration of the mission for two main reasons.

The first advantage is the classification and reduction of needs. Indeed some needs are present multiple times in the mission. For instance, the provision of thrust is a need present across the mission. Considering the function "provide thrust" classifies all these needs under one definition which can then be accommodated to all operational requirements. The second advantage of using a functional breakdown is to decouple requirements. If the mission implies both a thrust function and a cabin pressurization function, the requirements of each function are independent from the other. As a result the definition of each function directly implied by the mission can be defined independently from one another.

As a result the first step of the method proposed for the architecture requirement formulation is to perform a functional breakdown of the mission. This application of functional analysis is not new and its advantages supported by the sources sited previously in chapter 3 (c.f. functional analysis section). As a result, no hypothesis is implied by this aspect of the thesis.

4.2.1.2 Definition of Architecture Level Objectives

The mission is not only defined by non-negotiable tasks. Whether it is explicit or implicit, the mission always includes some form of figure or merits. For instance the figure of merit for a commercial aircraft development could be the net present value of the development project. At a lower level the figure of merit could be the revenue per passenger mile and/or the operational range.

4.2.1.3 SysML Representation

To declare the functional breakdown the SysML block definition diagram is used. The aircraft mission is defined by a SysML block. Each boundary function is represented by an out-bound flow port. In order to illustrate this implementation, let us consider a simple mission constituted of two functions: provide thrust and provide hydraulic power for flight control.

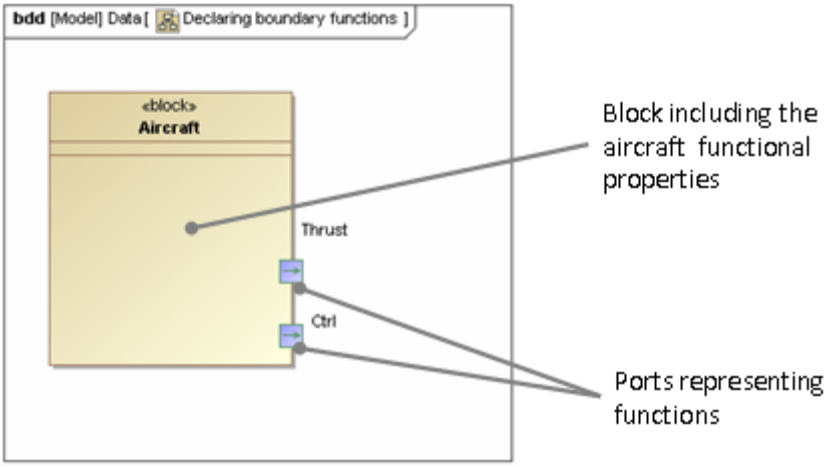


Figure 81: Implementation of functional breakdown

In this context, a block is a generic SysML element. The flow port represents some form of an interface allowing a flow (i.e. exchange) across the boundary of the element. In this case, the "flow" represents a functional capability that the aircraft (represented by the block) is expected to provide. Hence, the aircraft function is represented as a flow crossing the boundary of the aircraft block (boundary functions).

In is important to note that the functional breakdown of the mission is generic for any architecture concepts. It will be used to characterize the functional boundaries of the concepts defined in the architecture definition phase (activity 2).

4.2.2 **Mission Parameterization**

If the functional analysis provides a convenient breakdown for the mission needs, it is necessary to formulate the requirements implied by each function in a quantitative manner to constrain the sizing of subsystems. Depending on what the aircraft is doing (its flight phase, the ambient conditions, internal failures of its subsystems, flight and cabin crew inputs…), the functional requirements will be changing.

In order to capture these changes, it is necessary to define some basic vocabulary which will be used in this document. The requirements (which quantify the needs) are defined by the operating condition. Each operating condition implied by the mission will

be referred to as an "**operating scenario**" (or "**scenario**"). The ensemble of all operating scenarios will be referred to as the "**mission envelope**". For each scenario, a specific set of requirements is imposed on the architecture. The sets corresponding to each operating scenario will be referred to as "**functional requirement profiles**" (or "**functional profiles**").

Therefore, in order to accurately capture the constraints imposed by functional requirements, it is necessary to identify all possible scenarios. In order to ensure that the exploration is complete, a parameterization of the mission envelope is necessary. In order to frame and parameterize the mission envelope, dimensions must be defined. In order to serve these objectives, the **mission parameters** identified were: The flight phase, the flight conditions and the criticality. These mission parameters will be used to define scenarios.

4.2.2.1 Flight Phase

The **flight phase** refers to a segment of the mission which implicitly requires the performance of some set of functions. As a result, flight phases can be used to characterize either change in amplitude of some requirements or the inclusion or exclusion of functionalities (depending on their necessity in the phase). For instance, during phases on the ground the landing gear steering functions are enforced but deactivated once airborne. By observing what is implicitly required by the mission in each of its phase, it is possible to derive the sets of functionalities required from the architecture.

The flight phase influences the amplitude of the functional requirements, for example the value of thrust requirement in the take-off phase exceeds the one cruise. Therefore if the flight phase does not always fully determine the amplitude of requirements it is certainly an important contributor to its definition.

4.2.2.2 Flight Conditions

The temperature, pressure, humidity, altitude or speed at which the vehicle is operating will be varying over a range of conditions. Each of these operational properties may or must be considered. Each of them will have an influence on the amplitude of the functional specifications for any given scenario. For instance, a given thrust requirement value for an air breathing engine will be more stringent for a hot day compared to a cold day. Similarly the **flight conditions** may influence the amplitude of the functional requirement. This can be illustrated by comparing a day with icing conditions which imposes a requirement on the anti-icing subsystem compared to a "blue-sky" (i.e. normal) operation which requires little or none.

4.2.2.3 Criticality level of the Failure Configuration

An aircraft is defined to operate optimally with all subsystems operating normally, but it is also expected to survive in situations of failures. From the difference between the meaning of the terms "operating optimally" and "surviving", we can see that the degree failure in which the aircraft is operating influences the requirements imposed by the mission. For instance, let us consider a situation with complete engine shutdown where the survival energy must be provided from a Ram Air Turbine. Then any functional requirements not dedicated to recovering from the failure or survival of the passengers and crew can be eliminated if necessary. This degradation is justifiable for two reasons. The first is related to the survival of the aircraft in case of subsystem failure. If a critical subsystem has failed, priorities must be set amongst boundary functions. The second argument is related to performance. Most power subsystems have a redundant counterpart. But in order to limit the over-sizing of subsystems with respect to their normal operation, some degree of degradation is allowed when one of the subsystems fails.

From this observation, we can see that, in order to determine the functional profile implied by a given scenario, it is necessary to know what has failed and how this failure

allows us to degrade requirements. It is for this reason that we introduce the concept of **criticality levels (criticality depth)**.

In order to understand how the criticality of a failure influences the functional profile, it is necessary to understand the notion of **voluntary functional degradation**. Voluntary degradation occurs when the architecture operates in failure configurations. The term of "**failure configuration**" will be use to refer to the operation of the architecture with a specific combination of failed subsystems. In this work we shall consider two general types of operation. The first type corresponds to "normal" operation where all subsystems are able to operate normally (no failure). Normal operations are expected to meet the normal operation requirements (i.e. no **voluntary functional degradation**). The second type corresponds to the "failed" operations. They correspond to scenarios where one or more subsystems have failed. For some of failed operations, degradation in functional capability is acceptable. The criticality depth is defined on a scale going from zero to some maximum value. A shallow criticality (depth = 0) would therefore refer to scenarios where the architecture is operating in a normal configuration. The criticality of a failure is appreciated based on its probability to occur. Hence, a deep criticality would refer to scenarios resulting from a "highly unlikely" failure event. This low probability of occurrence gives license to the designer to severely degrade the functional requirements, therefore relaxing sizing constraints. On the other hand, if a subsystem is likely to fail the designer can not justify degrading the requirements.

In early conceptual phases, defining the probability of a failure event to occur requires insight from the architecting team. The depth of criticality assigned to a failure scenario is in fact the result of a trade-off and should almost be considered as one of the degrees of freedom in the architecture design problem. This tradeoff can be illustrated by the following example. Let us consider an architecture with a large electric load (e.g. electric ECS). In this example we assume that the ECS is energized by two electric generators (see architecture presented on the left hand side of Figure 82). Since these

generators are “redundant” their sizing requires the consideration of the situation where one of the two generators is failed. This failure leads to the consideration of a failure configuration (see right hand side of Figure 82). The depth under which this failure is classified will then define by how much the designer can allow the functionality to be degraded.

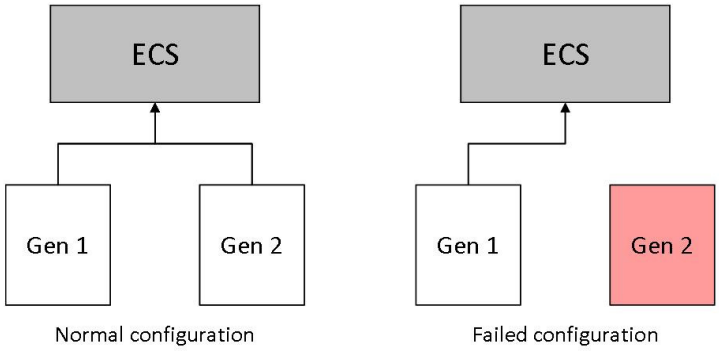


Figure 82: Criticality example

If criticality depth = 0 is chosen, then the requirements imposed on the failed configuration are the same as for a normal configuration (with both generators operating). In this situation we can say that the two generators are truly redundant. However, the cost of this redundancy will be that each generator will be 100% oversized for their normal operation.

If some criticality depth is assigned to the failure, the assumption is that the failure will be sufficiently unlikely to accept some reduction of the ECS functionality when either generator fails. In this situation, we can not say that the generators are truly redundant, but rather, the over-sizing is limited thanks to this reduction in requirements.

Beyond the limitation of over-sizing, the use of degradation should not be abused since it has an important hidden cost. If the generator failures are considered as highly critical, the sizing constraint is relaxed, but the reliability requirement on its design is now increased. Hence, what really counts is the ability of the architecture to deliver energy to the load (in this case the ECS). Among the specifications qualifying on this functionality, there are either safety rules (certification) or dispatch reliability targets

limiting the frequency at which the functionality can be either degraded or eliminated. Therefore, the deeper the failure criticality is, the higher the reliability target at subsystem level.

In order to simplify the usage of the concept of criticality a scale is proposed and presented in Table 13. The degradation scheme proposed in this scale was inspired from the certification authorities failure classification [73]

Table 13: Criticality level description

	Description of applicable failures	Degradation scheme	Example
Normal	No failure	None	-
Level I	Failure configuration which should not stop the aircraft from getting destination (dispatchable failure)	Slight degradation of non-vital functionalities dedicated passenger comfort is acceptable. Degradation should be transparent on aircraft performance	- Single generator failure - Failure of an ECS pac
Level II	Failure of (a) critical element(s). This failure leads to aborting the mission	Elimination of most non-vital functionalities dedicated to passengers is acceptable. Slight degradation of aircraft is acceptable.	- Failure affecting less than 50% of propulsive capability
Level III	Failure threatening the survival of the aircraft. This failure leads to aborting the mission.	Elimination of all non-vital functionalities. Significant passenger discomfort is acceptable. Temporary but significant degradation of aircraft performance is temporarily acceptable.	- Loss of power plant - Total loss of propulsive power

This criticality level description and associated degradation schemes are provided as an example for the methodology proposed. In practice, they must be tailored to the mission under consideration. But since the criticality levels are used to define the functional requirements, it is necessary to define the degradation scheme applicable to the situation.

4.2.3 **Requirement Quantification**

In the beginning of this chapter the following research questions was formulated:

Research Question 4: How can we characterize the mission in a fashion that captures all requirements driving subsystems sizing?

Three types of mission parameters were presented earlier: **flight phase**, **flight conditions** and **criticality depth**. These factors decompose the operational variations influencing the functional requirements. Together these functional requirements define

the functional profile associated with the scenario. Using these properties, operational scenarios can be defined by establishing their "coordinates" with respect to these three "dimensions" in the mission envelope.

4.2.3.1 Overview of the Method for Scenario and Functional Profile Definition

The **functional profile** corresponding to each scenario can be redefined by modeling the influence of the three parameters discussed previously on the functional requirements. Based on this observation and in an attempt to the address the research question above, the following hypothesis is proposed:

Hypothesis 1: The functional requirements can be characterized in a complete, practical and accurate fashion by defining scenarios based on flight phases, flight conditions and criticality depths.

Based on this approach, a scenario is defined using the flight phase, flight condition and criticality level. The functional profile associated can now be defined based on the following notional equation:

$$\textit{Functional Profile.} = f(\textit{flight phase, flight conditions, criticality level}) \quad \textbf{(14)}$$

This formula has two advantages. The first and most important is related to the definition of the functional requirements and specifications. If the factors are independent of the architecture concept, the specifications assigned to functions can be defined in a way which is truly independent from design choices. Also the assumptions used in the definition of functional specifications are traceable since they are clearly associated with an operational context. The other great advantage is the facility with which new scenarios can be defined using these parameters. Indeed if equation (14) is defined for each function implied by the mission, by simply specifying a scenario with those three parameters, all requirements associated with it can be accessed.

4.2.3.2 Modeling Scenarios in SysML

In phase 2 the architecting team will use the functional profiles to specify scenarios. Based on hypothesis 1, scenarios can be described by specifying its flight phase, flight condition and criticality level. In order to facilitate the scenario definition process a type of SysML element is created. This type is defined by a SysML stereotype. As will be described later in this chapter, the scenarios will be defined using a SysML activity diagram. The activity diagram is composed of "activities". Therefore the stereotype definition is a specialization from the SysML activity type. It is specialized by the addition of three tags. The tags define pieces of information which are necessary to complete the definition of the element. Therefore three tags are created to characterize the scenario with respect to its operating phase, condition and criticality.

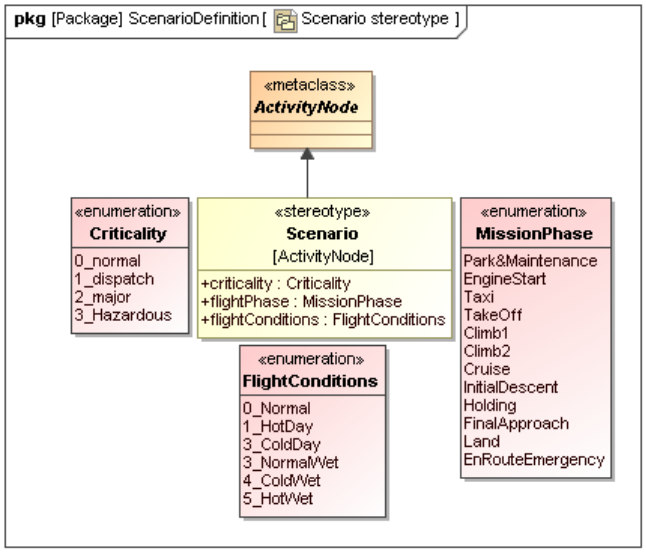


Figure 83: Definition of the scenario stereotype

Figure 83 provides an example of a scenario stereotype definition. This definition is defined in a package diagram (PKG). In this definition the tags are defined as enumerations. Using enumerations ensures that the definition of the scenario is valid.

4.2.4 **Conclusion**

In this section, we have shown how the aircraft architect can frame the mission and organize the requirements associated with it. With this approach, an architecture-independent formulation is defined. This formulation can then be used to guide the architecture concept definition and the sizing of subsystems.

4.3 Subsystem Knowledge Preparation

Figure 84: Process overview - Subsystem knowledge preparation

The purpose of this activity is to prepare modeling bricks with standardized interfaces. Each of these modeling bricks represents a subsystem. It contains the "knowledge" necessary to size and analyze the subsystem in its architectural context. The interfaces of these models are standardized to facilitate their automated integration necessary analyze the architecture. The process implied by this activity is defined around three tasks:

- Task 1: Identification and classification of subsystem candidates
- Task 2: Definition of standardized modeling interfaces
- Task 3: Packaging design knowledge for subsystem sizing

The tasks implied by the preparation of subsystem knowledge are performed by the subsystem experts. The methods supporting each these tasks are presented in this section.

4.3.1 **Identification and Classification of Subsystem Candidates**

This task consists in defining a library of subsystems. This library will list the subsystems which will be modeled in task 3 (Subsystem sizing models). These subsystems will also provide the blocks necessary to represent the architecture concepts in the architecture concept definition phase. The construction of this library is organized around three subtasks:

- The first subtask consists in identifying the subsystems candidates for integration
- The second subtask will require the expert to analyze its subsystem functionally
- The third subtask verifies that all the necessary subsystems were included in the library

The library is defined in SysML. The purpose of this library is two fold. In the setup phase the SysML diagrams guide the definition of subsystem models. In the architecture concept definition phase, these diagrams will guide the definition of the architecture.

4.3.1.1 Identification of Subsystem Types

The subsystem modeling block approach is based on three important assumptions:

- Each subsystem can be designed in a generic fashion (i.e. for a given subsystem concept, the same design approach can be used regardless of its architectural context)
- Each subsystem can be sized independently from the other, granted that its context of integration is described properly.
- The context of integration can be described in a generic fashion with a finite number of design variables

These three assumptions must guide the AT in the definition of the level of granularity used in the study. If the subsystems used are too large these assumptions will

be respected only for specific family of architecture concepts. As a result the type of trades that can be considered will be limited. As the level of granularity becomes thinner, the number of subsystems increases and the architecture model more complex; but each subsystem becomes smaller and functionally simpler. As a result the assumptions listed above are valid for an increasingly larger scope of concepts.

In this subtask the objective to identify the ensemble of subsystem candidate. In this context the term "candidate" implies that the subsystem is considered for integration in the architecture. Therefore, the list of subsystems must be defined in accordance with the level of granularity previously defined and should include all subsystem alternatives deemed necessary to define architecture alternatives. The term "necessary" will be defined more precisely in paragraph 4.3.1.3 of this section. Subsystems alternatives must be defined when multiple technologies or subsystems concepts can be considered for performing the same function.

Subsystem alternatives should be defined when considering different types of subsystems. The term "**type of subsystems**" (or subsystem type) has a very specific meaning in this context. Herein, any subsystems falling under the same type will have to be sizeable by the same model. Therefore, the fundamental rule is that, if two subsystems are sufficiently similar to be represented by the same model, it then makes sense to use only one type to capture both. Two subsystems share the same type, if they are the result the same design framework and share similar functional interfaces. For instance, one can assume that an 80klb turbofan and 35klb turbofan share their "type" because:

- They both result from a similar design framework
- They share similar functional interfaces (they both can provide thrust and require fuel to operate)

On the other hand, multiple alternatives should be defined for subsystems including different technologies as each technology will imply a different pool of knowledge and therefore a different sizing model. Such a situation can be illustrated by

the differences between electric and mechanical pumps. These pumps may be similar in their nature, but they rely on different technologies and therefore on different design knowledge. It is also necessary to identity multiple subsystem types when the functional interfaces between subsystems are different. In this situation, the functions induced by these pumps are different. One pump induces mechanical power needs when the other induces electric needs. Consequently, these pumps will require interfaces with their energy providers. Therefore, for both conceptual and modeling reasons it is necessary to represent them under different types.

Similarly, subsystems with drastically different concepts (i.e. internal architecture) should not be listed as the same subsystem. To illustrate this situation one may consider a turbojet and a turbofan. One may argue that they are based on the same technology (air-breathing turbo machinery) but used with different design settings. Since the analyses required for these two concepts are drastically different, it is more practical to represent them as two different subsystems types in the library. This will imply that two different subsystem models will need to be defined.

To conclude on the description of this task, it is difficult to generalize on how subsystems should be classified since it is really up to those defining the models. However, it is essential to maintain a strict functional format for each subsystem type as this format will eventually become both the leading thread for the architecture definition and the keystone for the model builder.

4.3.1.2 Functional Analysis of Subsystem Types

4.3.1.2.1 Definition of Functional Analysis

In this context the functional analysis will consist in identifying the functional interfaces between the subsystems and their environment. The result of the functional analysis of a subsystem type is defined as its "**functional description**".

A subsystem can interface with its environment either by providing a function or by imposing a requirement. To illustrate this concept, let us consider an electric generator. The electric generator provides electric power if granted mechanical energy. Therefore one can say that it has two functional interfaces. The first identifies the capability of the generator to provide electricity, the other the mechanical requirement necessary to perform its function.

For each subsystem type identified in the previous task, its functional analysis must be performed and its capability and induced requirement identified. This information gives an overview on how the subsystem must be integrated in the architecture.

4.3.1.2.2 SysML Implementation

In order to facilitate the recording of subsystem types and their functional description, an implementation process in SysML is proposed. This task implies creating a library in SysML. A library is defined by creating a modeling package. This library is itself composed of two sub-packages. The first will list the subsystem types and the second the types of functional flow. Function flows and subsystem types are defined using block definition diagrams.

Coming back to the example where we are investigating architectures for thrust and control, let us assume that the subsystem types chosen by the experts were: a turbojet, a fuel system, an electric generator, a mechanical pump and an electrical pump.

By performing a functional analysis on these subsystems, the experts concluded that the turbojet has two capabilities (provide thrust and mechanical power) and one requirement (provide fuel). Similarly the electrical generator provides electric power if supplied with mechanical energy, etc… Once all subsystems were analyzed functionally, five functions can potentially be exchanged:

- provide fuel
- provide thrust
- provide electric power
- provide mechanical power
- provide hydraulic power

The first step in the SysML implementation is to identify these function flows (i.e. functional exchange). Function flows are defined as SysML blocks stored under function flows (see example in Figure 85).

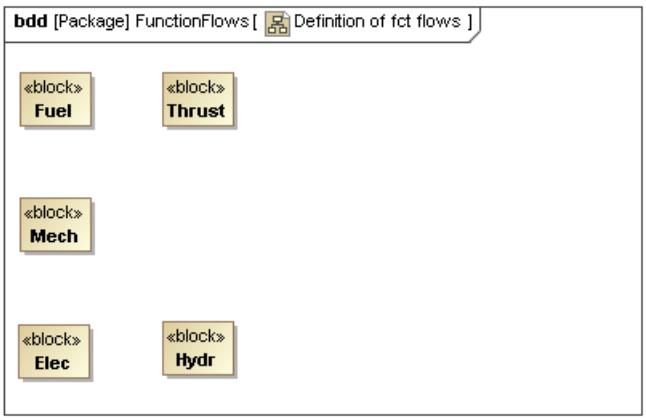


Figure 85: Identification of function flows

Once function flows are declared, subsystem types can be defined. The types are defined as blocks and the functional interfaces are defined as flow ports. The functional interfaces are defined via two elements. The first element is the directionality. If the functional interface is a capability the port is defined as being outbound. If the functional interface is a requirement the port is inbound. The second element is the type of the functional interface. To define a type, the port is allocated to a function flow. Figure 86

provides an example for a functional definition of subsystem types. In this example, the ports are green-arrowed boxes on the boundary of the blocks representing each subsystem.

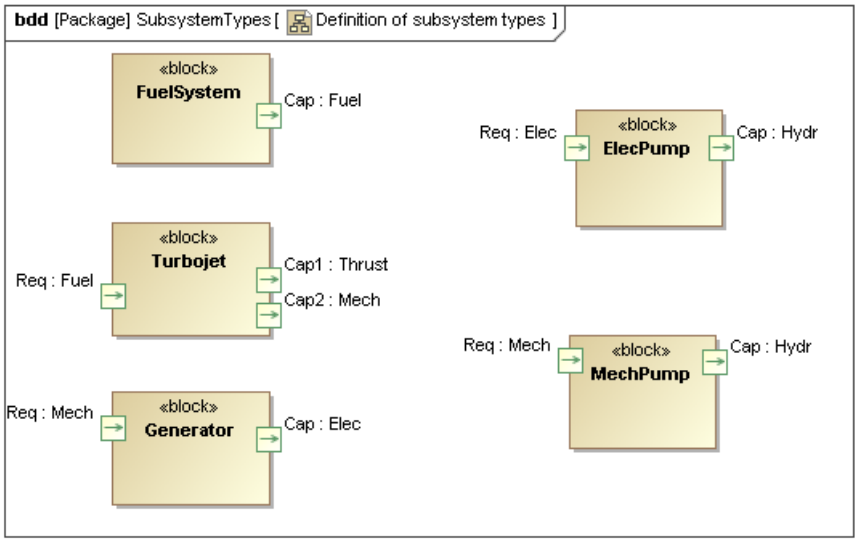


Figure 86: Functional definition of functional types

4.3.1.3 Verifying Completeness of the Subsystems Library

A list of subsystem types is complete once the following two conditions are met:

- All subsystem concepts that the architecting team wants to consider for trade-off are captured by one of the subsystem types
- Each functional flow is assigned at least once on a capability port

The first condition ensures that the design space for architecture trade-offs was fully defined and that all concepts of interest can be represented by the elements composing the library. The second condition prevents situations where an induced function can not be fulfilled because one of the subsystems necessary to the architecture was left out.

4.3.2 Negotiation of Modeling Interfaces

In order to automate the integration process of subsystem models, it is necessary to have standardized interfaces between subsystem models. This section will introduce a method relating the functional relationship between subsystems and the flow information

relating their models. Based on this methodology, flows of information are standardized along with model interfaces.

Hence, the description of the description of this task is described in two subsections. The first subsection will describe the method used to standardize the exchange of information (Definition of Functional Flows). The next subsection will focus on the interfaces of subsystem sizing models as implied by the functional flows.

4.3.2.1 Definition of Functional Flows

Let us consider the design problem of the subsystems in the two-element architecture presented in Figure 87. The two subsystems are related by a function ("provide mechanical energy"). In the design process of each subsystem, this relationship must be considered. For instance, the amount of mechanical power required by the pump is going to influence the design of the engine, in the same fashion the characteristics of the mechanical power delivered by the engine (e.g. RPM) will constrain the design of the pump. From this example, we can observe that there is a mapping between the functional relationships and information necessary to the subsystem design problems.

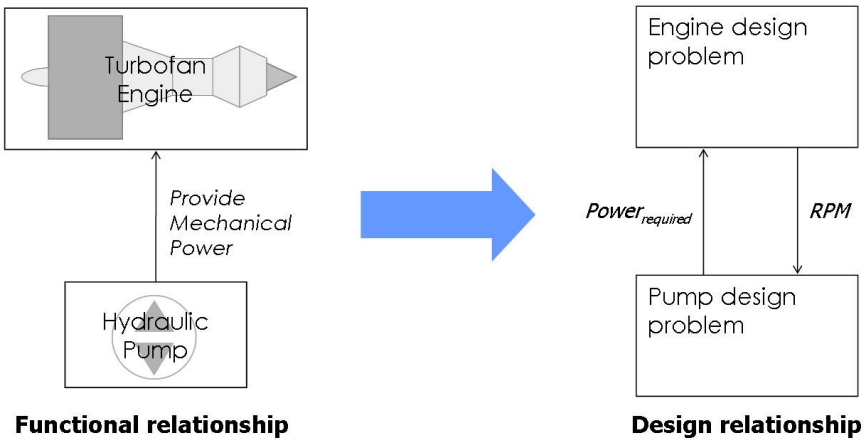


Figure 87: Parallel between functional relationships and sizing relationships

From the description of activity 1b-2, we saw that the sizing process is an approximation of the design problem. Hence, I will be assuming that the information

necessary to the sizing is included in the information necessary to the design. Therefore, putting these observations together allows me to formulate the following hypothesis:

Hypothesis 2.1: The functional relationships between two subsystems characterize the flow of information between their sizing processes.

When the architecture is defined, the functional relationships must be identified. By hypothesizing on a parallel between subsystem functional relationships and the flow of variables, a useful parallel between the description of the architecture and the numerical model describing it is created.

4.3.2.1.1 Organization of Functional Flows

In this thesis the term **functional flow** will refer to a functional relationship characterized by a directed exchange of information (or flow of variables). In order to make a generalization on functional flow, let us consider the following functional relationship between subsystem A and B. In this relationship, subsystem A performs function X for subsystem B.

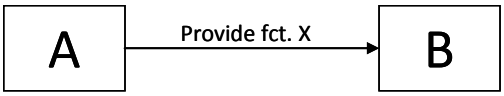


Figure 88: Notional functional relationship

In order to characterize the functional relationship, several pieces of information may be necessary. This information can originate from or flow to three possible locations:

- The model defining the functional source (here subsystem A)
- The model defining the functional load (here subsystem B)
- The architecture concept definition

The model for the subsystem will define or receive operational constraints. These constraints drive the size of the subsystem. The architecture concept definition is defined before the execution of the model. It specifies information but does not receive any

during the sizing process. Based on these observations, a functional relationship may imply three possible flows of variables between three modeling elements. These flows are notionally presented in the following graph:

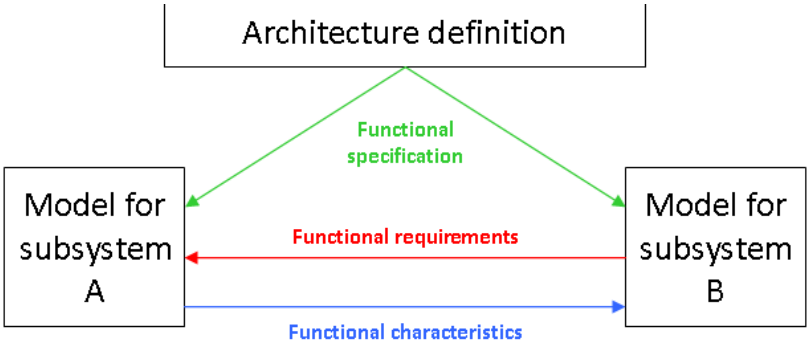


Figure 89: Classification of variable flows

4.3.2.1.2 Characterization of Functional Flows

In order to store the definition of the functional flow, each flow is defined as a SysML block. This block is defined as being composed of three parts corresponding to the three modeling elements presented above: the functional source, the functional load and the architecture definition (here defined as "architectureConcept"). Each modeling element has two ports. One represents the output of the model element (variables emitted by the model), and the other represents the input (variables received by this model elements). The definition of the generic block for function flows is defined in a BDD represented in Figure 90.

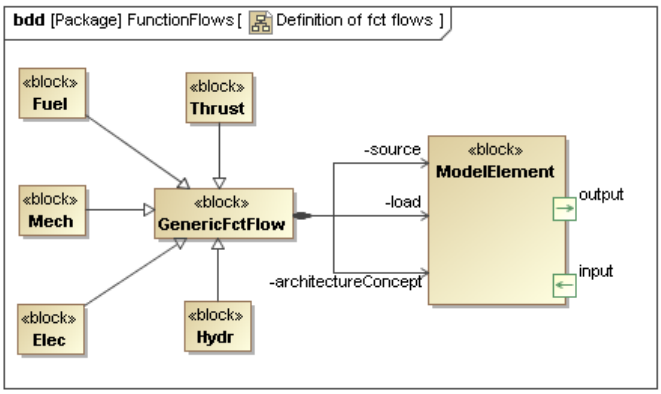


Figure 90: Definition of the generic block for function flow

This generic functional flow is then used to specify the function flow representing each of the functions present in the architecture and identified in activity 1b-1 (as part of the functional analysis of subsystem types). Therefore, the graph represented above is in fact a completion of the graph which was presented in Figure 85. In Figure 90, we can see that the functions identified previously are now specializations of the generic function flow block. This specialization allows them to inherit the three modeling element structure (source/load/architecture concept).

For each block representing a function flow, an internal block diagram (IBD) must be defined. This IBD represents the flow of information necessary to characterize the function (**functional flow**). This information is defined by drawing an item flow going from and to the relevant model elements. To illustrate this task, the characterization of the electric power function is presented in Figure 91.

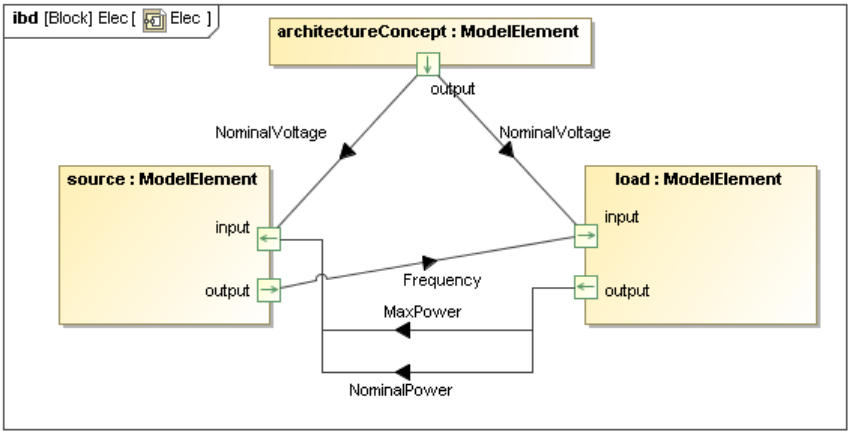


Figure 91: Characterization of variable flow with an IBD (example 1)

This IBD describes the variable flows. The name of the flow (represented next to the arrowed lines connecting the model elements) describes the type of information carried by the flow. In this example there are two functional requirements. One represents the max power requirement for the scenario and the other the nominal power requirement. There are two functional specifications, both of them defining the nominal voltage expected from the network; one provides the information to the model sizing the source, the other to the model sizing the load. Finally there is one functional characteristic which

defines the conditions at which the source is providing the function to the load (in this example this functional characteristic is the current frequency).

Each of the items conveyed by the flows are SysML blocks which contain value properties. These value properties define the generic name of the variables of the sizing model. This way, once an architecture concept is defined, the variable connections representing the functional relationships can be created automatically.

4.3.2.2 Modeling Interfaces of Subsystem Sizing Models

In this section we have described how the sizing models have been defined and how they are expected to interface. Then if we use the analogy of a puzzle for the model representing the architecture, in this section we have described how:

- The bricks (or elements) of the puzzle are defined by describing the sizing process
- The interfaces of the bricks are standardized by describing the flow of information associated with each function.

We will now consider, via a simple example, how the functional standards described previously are both specifying the flow of variables received (and that must be produced) by each subsystem model, and how this flow of variables will be defined depending on each scenario.

4.3.2.2.1 Standardization of Interfaces

Earlier we saw that the architecting team will define the flow of information implied by each type of function. We have also seen that each subsystem type is defined functionally (identification of the function the subsystem is capable of providing and the function it induces). By associating these pieces of information, a minimal set of inputs and outputs is defined, In order to illustrate this important point, we shall consider an example. Let us assume that the two function flow types were defined as described by Figure 92. Using these function types, two subsystem types are defined in Figure 93.

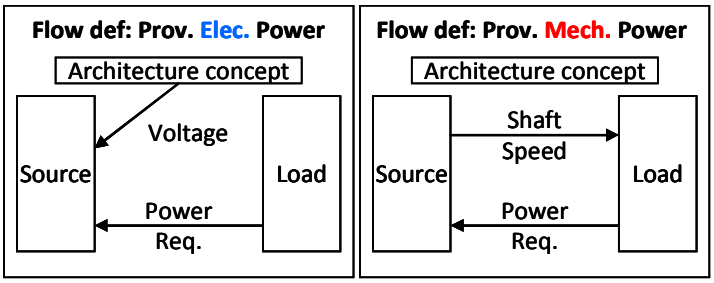


Figure 92: Example of function flow types

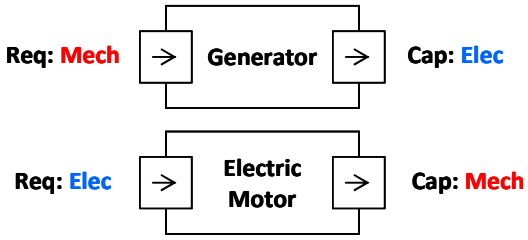


Figure 93: Example of subsystem types

Based on these definitions, a minimal set of variable interfaces is defined for the generator and the electric motor sizing models. Since the generator is receiving mechanical power, it is expected to produce the one variable quantifying the mechanical power required (as described in Figure 92). It will also receive one variable for the speed at which its shaft is expected to operate. Using the same logic on its electric functional flow, the generator will receive two variable: one for the voltage of the current to deliver, and the other one quantifying the amount of power that must be delivered. Therefore based on Figure 92 and Figure 93, the following set of variable interfaces is defined.

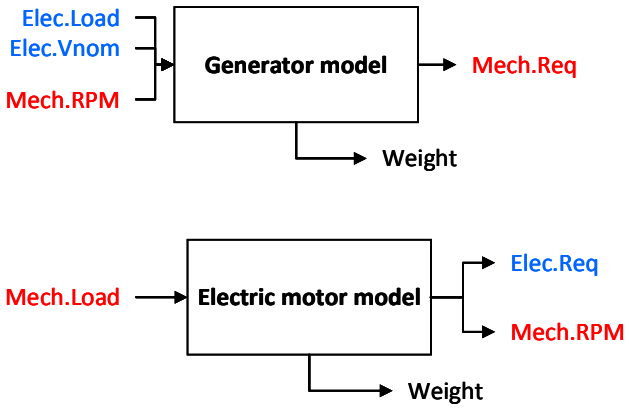


Figure 94: Example with generator and motor model interfaces

Based on this example we can see that great care must be dedicated to both the definition of the variable flows of functions and the functional analysis of subsystems. It is important that the functions that are assigned to the model will provide all the information necessary to the sizing of the subsystem. It is also important to consider that different subsystems may require different information from a given functional relationship. Therefore, it is important that the characterization of the functional flow captures the information necessary to all subsystem. Based on this observation, we can see that the negotiation process within the architecting team fulfills this role. Once agreement is reached it is essential that the subsystem sizing models are compliant with the flow standards.

4.3.2.2.2 Connections Scenario by Scenario

In activity 1a, we have observed that the sizing of the subsystem may be constrained differently depending on the **operational scenario**. In order to accommodate this multiplicity in requirements and operating conditions, the variables implied by the functional flow are defined as arrays. Each element within the array represents the requirements (or specifications) associated with a specific operating scenario. In order to illustrate this format, let us consider a simple architecture composed of one generator (Gen1) and two motors (M1 and M2). In scenario 1 Gen1 provides M1 and in scenario 2, Gen1 provides M2 (see Figure 95).

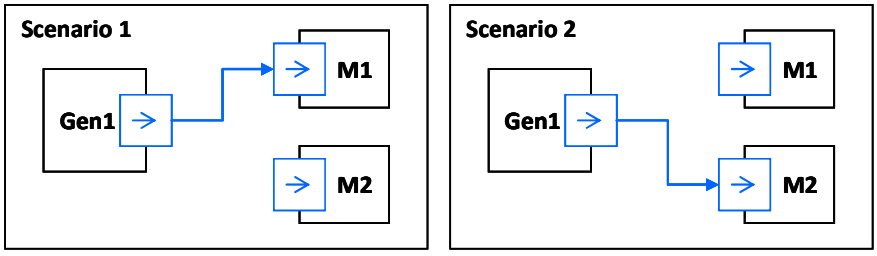


Figure 95: Different requirements in different scenarios

In this situation, the connections are defined for each of the element within the arrays. Hence for the first scenario, the value of the load on Gen1 will be defined based

on the requirement expressed by M1. Similarly, the second element representing the scenario 2 will be defined by relating the value generated by M2 and the value received by Gen1. These connections are represented graphically in Figure 93.

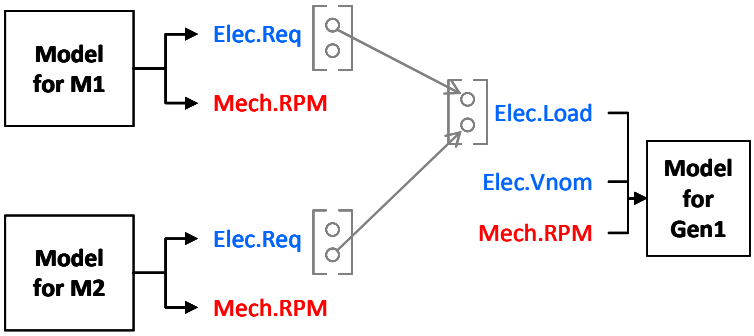


Figure 96: Example of variable connections in array

Note: it is important to observe that the variables characterizing the functional relationships are arrays. Each rank in these arrays will qualify same operational scenario. The size of all arrays is defined by the total number of scenarios considered in the mission.

4.3.2.3 Conclusions about Modeling Interfaces

In this task we have described how the flow of information can be standardized by considering the functional relationships it represents. This standardization allows for the definition of generic interfaces that can be used for the automated integration of the subsystem modeling bricks. Now that we have observed how the interfaces are defined, we shall now consider how these modeling bricks (or subsystem sizing models) are defined.

4.3.3 **Packaging Design Knowledge for Subsystem Sizing**

The modeling approach used in this thesis is a bottom-up approach. This approach is based on models capturing the attributes of sized subsystems and integrating them at architecture level. In this section, methods are formulated to structure the knowledge necessary to estimate the attributes of subsystems. This knowledge is provided by the subsystem experts. They provide this knowledge in the form of subsystem sizing models. The presentation of this method will first be based on a theoretical analysis of sizing which will be concluded by its underlying hypothesis. This theoretical introduction will be followed by a critical analysis of different implementation approaches to this method.

4.3.3.1 What is Sizing?

Sizing is an estimation of the major attributes of a finalized system, based on the requirements imposed on it. The term finalized system refers to the outcome of the design process. We can therefore conclude that the sizing is an approximation of design. In Chapter 2 we have defined the design process as:

> Design is the process of creating the "best" physical solution possible which can fulfil a set of requirements.

Based on this definition we can say that design can be formulated as the following optimization process:

Maximize "Goodness of the solution" *subject to* the fact that design must be at least able to do what it is supposed to do.

In Chapter 2 we observed that:

- The negotiable needs, which map to the formulation of the "goodness of the solution" (i.e. value), can be defined by design objectives
- The non negotiable needs, which define "what the solution must, at least, be able to do" (i.e. its capability), are defined by functional requirements (Ri).

Based on these observations, the subsystem design problem can be formulated as follows:

$$\underset{X_i}{Max} \quad V_{system}(Att_i) \tag{15}$$

Subject to:

- $R_i \leq C_i(X_i, Spe_i)$
- $Att_i = f(X_i, Spe_i)$

When we design a subsystem, the theoretical objective is to provide the subsystem which will optimize the value of the system overall. Since the subsystem is (by definition) an element of the system, the value of the system as a whole is function of the attribute of the subsystem (Att_i). The attribute of the subsystem are themselves a function of X_i (subsystem design variables) and Spe_i (the operating specifications. Given this formulation of the design problem, and the fact that the sizing process is an approximation of the design process, we deduce that:

Hypothesis 3.1: The sizing process is an approximation of an optimization process.

This hypothesis partly addresses *Research Question 3*:

Research Question 3: How can we mathematically formalize subsystem sizing while allowing for their coordinated optimization?

There are two aspects to this research question. The first concerns the sizing of the subsystem while the second refers to the global optimization of subsystems. In order to formulate a solution which addresses the search question above, it will be broken down in two research questions. Each of these questions focuses on a specific aspect of Research Question 3.

Research Question 3.1: How do we mathematically formalize subsystem sizing

Research Question 3.2: How do we size the subsystem in a fashion that captures the ability to adapt the subsystem solution to the global optimization of the architecture?

Hypothesis 3.1 provides an answer to *Research Question 3.1*. [1]

4.3.3.2 Different Forms of Approximation:

Based on this hypothesis, the sizing model provided by the expert will provide a means to estimate explicitly or implicitly the optimal value to the optimization problem corresponding to its subsystem design problem. In order to adapt to different subsystem specificities or context of integration, different sizing methods are proposed. The following two approaches are introduced:

- Functional regression
- Optimizer-based sizing.

Both these methods are reviewed in the following paragraph. As part of their description their implicit assumptions are reviewed in order to guide the reader with regards to the best approach to their problem.

4.3.3.2.1 Functional Regression

Functional regressions are certainly the common approach to sizing in conceptual design. This approach attempts to relate the functional capability to the volume or weight of a system. Models based on "power density" or "specific power" are examples of this approach in its simplest form. When these sizing models are used, we are taking into account the functional requirement. Then, as soon as one of the constraints $R_i \leq C_i(X_i, Spe_i)$ is satisfied, the model will assume that the optimality of the design

[1] We will come back to *Research Question 3.2* in section 1.1

problem has been reached. The advantage of this method is certainly its simplicity as it can be formulated since an explicit equation:

$$Att_i = f(R_i, Spe_i) \quad \textbf{(16)}$$

But if we compare the terms present in equations (15) and (16), we can see that the notion of value has disappeared. This absence therefore implies at least one of the following assumptions:

- The design is simple enough to assume that there is only one or few similar alternatives that can fulfill the functional requirement
- The design space is highly constrained and there is only one alternative that can fulfill the functional requirement
- The model was constructed on assumptions which guaranty the optimality of the subsystem for its mission within the architecture

Assumptions 1 and 2 are similar in their nature but different in their cause. An example for assumption 1 would be the sizing of an electric cable. Considering the fact that an electric cable is a wire embedded in a protecting sleeve, we can assume that the weight, cost or energy losses (attributes) of the cable will be sized as a function of the current it must carry, and its required length (functional requirements).

Assumption 2 refers to specific situations where little knowledge or design freedom is available in the design of a subsystem. For example, if the designer must use the products from one supplier to purchase the hydraulic pump for the architecture. If the supplier has only one pump per flow capacity rating, then no trade-offs may occur and the designer can only consider the one that "does the job". In this situation, the lack of design freedom reduced the design problem to a constraint matching problem rather than an optimization problem. But even when the choice is limited to off-the-shelf alternatives, there can always be several alternatives with the ability to do the same thing. But one of them will be most appropriate for the architecture than the others.

Even with conceptually simple systems (e.g. the electric cable) the assumptions above may be contested. If one starts wondering if it would not be preferable to consider a more efficient but thicker (heavier) cable, this type of model will not support this type of trade-off (see assumption 3).

Therefore, we can conclude that the functional regression method provides a simple approach to sizing, but fails to consider subsystem-level trades. Consequently, this solution does not provide a satisfactory solution to *Research Question 3.2* which requires the ability to adapt the subsystem sized solution in support for the architecture optimization. In the conceptual analysis of an aircraft system architecture, using this modeling approach to sizing may lead to inappropriate (suboptimal) subsystem estimation and provide incomplete guidance to the definition of the design problem. On the other hand, its mathematical simplicity makes it easy to setup and allows for rapid sizing estimations.

4.3.3.2.2 Optimizer-Based Sizing

In order to improve the quality of the sizing process a new sizing approach is proposed. This approach attempts to use a formulation which is closer to the design problem it attempts to approximate. In order to do so, a sizing process based on an optimization setup is implemented. Maintaining this parallel between the design process and the sizing process captures a simulation of the actual design process. The optimizer-based sizing process is formulated mathematically as:

$$\underset{X_i}{Max} \quad V\left(Att_i, \Gamma_i\right) \tag{17}$$

Subject to:

- $R_i \le C_i\left(X_i, Spe_i\right)$
- $Att_i = f\left(X_i, Spe_i\right)$

Outcome: Optimal subsystem description: X_i^* and Att_i^*

Definition of the subsystem design variables X_i requires the use of an optimizer. In addition to the optimizer, relating the design variables to the subsystem attributes (Att_i) and capability (C_i), requires the use of analysis tools. The objective function $(V(Att_i, \Gamma_i))$ is a pondered sum of the subsystem attributes (Att_i) weighted by the factors contained in Γ_i (referred to as priority factors). The **priority factors** will give different degrees of importance to the attributes. Therefore as the values contained in Γ_i change, different subsystem solutions optimizing different attributes will be selected for representing the sizing process,

The assumptions implied by this approach are that:

- The analysis models are accurate in producing their estimation of the system attributes.
- The objectives quantifying the notion of value contribute to the optimization of the architecture.
- The design process will follow the same optimization objectives and functional specification as the sizing process.
- It is possible to automate the optimization process searching for X*.

The fact that this approach follows a format which is closer to the design process makes its assumption less restrictive in its prediction.

The availability of an analysis model is not a very constraining assumption. Nowadays, most experts conduct their developments based on numerical models which estimate design attributes based on design variables. Therefore, using this optimization approach enables the integration of specialized analysis tools.

The second and third assumptions imply that if the optimal, actual and sizing objectives are different the estimated attributes will not be accurate. In other words, if we are optimizing on objectives which are different from what is actually necessary for the

architecture, the sizing estimation will be conservative, and the design outcome suboptimal.

The fourth assumption implies that the optimization of the architecture will be able to find automatically the optimal design. The fact that the design optimization process is subjected to constraints (defined by functional specifications) increases the complexity of the convergence process necessary to identify the constrained optimal. Also, the run time associated with the convergence process may become prohibitive.

We can also observe that this optimization approach includes the design variables of the system under consideration. This can be considered both as an advantage and a disadvantage. If we consider the context of the Architecting Framework, the advantage of having access to the subsystem design variables within their sizing model is that it provides some degree of transparency on what is happening at subsystem level. On the other hand, given the fact that the trades that will be performed by the Architecting Framework are at the aircraft level, the inclusion of subsystem design variables will make the model "crowded" with subsystem design variables.

4.3.3.3 Discussion

This formulation of the sizing problem as an optimization problem relies on the fact that functional specifications are quantified and objectives defined. It means that in order to size the subsystem, its problem definition must be defined in a quantitative fashion. Doing so does not only allow for automating the sizing process of the subsystem. It also provides a platform enabling a numerical simulation of the design process of the subsystem. In this simulation, the input factors are the functional specifications and objective function parameters.

This simulation approach does not only provide the means to automate the sizing process. It also allows playing trades on the objectives at subsystem level. If we consider the simple example of the electric cable, trading between the weight of the wire and the

losses in transmission could not be considered by simple functional regression approach. Using the optimizer base sizing allow for trading off between different objectives. By launching different simulations representing different design instructions (i.e. subsystem objectives), the effect of the optimization priorities at subsystem level can be observed.

In this sizing approach the sized subsystem solutions can be adapted to architecture objectives. This adaptation is facilitated by the priority factors which define a relationship between the system-level and subsystem-level objectives. For this reason I believe that *Hypothesis 3.1*, formulated earlier in this section, addresses *Research Question 3.2*.

4.3.4 **Conclusion**

The methods presented in this section provided a means to organize subsystem-level knowledge in an architecture independent way. These methods attempt to provide a framework for the organization of knowledge that is both adaptable to the specificities of each subsystem, but standardized in order to be able to integrate this knowledge at the architecture-level. This standardization allows for the systematization of the definition of information flows associated with functions (functional flows). Also the sizing methods presented earlier ensure that the logic associated with subsystems models supports the general bottom-up analysis approach.

This section has presented how the modeling bricks were constructed and we also observed how the information related to the functional relationship (i.e. non-negotiable) between subsystems was addressed. In the following section, we shall observe how these modeling bricks will be integrated from an optimization point of view.

4.4 Sizing and Optimization Approach

This section will step away from the general process performed as part of the proposed framework (i.e. do not look for a "sizing and optimization approach" activity in Figure 79 presenting the overall process). This section shall present the general philosophy supporting the sizing approach presented in this thesis.

In the previous section, the sizing process was presented as an optimization problem. The optimization problem is constituted of constraints and an objective function. The constraints are qualified by functional specifications. The definition of these specifications was presented in the description of the modeling interface definition. The present section will now focus on the origin of the subsystem-level objective function. The challenge associated with capturing subsystem-level objectives is a key challenge in architecture design. This challenge was formulated previously by *Research Question 5: How do we translate fundamental objectives into subsystem objectives?*

In this thesis, a method is proposed to translate architecture-level objectives into subsystem-level objectives. This presentation is organized in three subsections. The first will lay the theoretical background necessary to understand the fundamental inspiration for the method. The second subsection will introduce a mathematical formulation of the method. This presentation is concluded by an observation of the opportunities offered by the coordinated optimization to formulate a design problem guiding post-conceptual subsystem developments.

4.4.1 **Theoretical Perspective on Architecture Design Activities**

In Chapter 2, conceptual design was described by the following equation:

$$\underset{X_0}{Max} \; V(Att)$$

Subject to:

- $R \le C(X_0, Att_\bullet, Spe)$
- $Att = f(X_0, Att_\bullet, Spe)$

(Subsystem design problem def.)

- $Spe_\bullet = f(X_0, Att_\bullet, Spe)$
- $R_\bullet = f(X_0, Att_\bullet, Spe)$
- $\Gamma_\bullet = f(X_0, Att_\bullet, Spe)$

(Resolution of subsystem design pb. – subsystem sizing)

- $Att_\bullet = f(\Gamma_\bullet, R_\bullet, Spe_\bullet)$

(3) Outcome:

- Optimal architecture description:

  X_0^* (Objective 1)
- Formulation of subsystem design pb $Spe_\bullet^*$, $R_\bullet^*$, and $\Gamma_\bullet^*$ (Objective 2 and 3)

In order to address *Research Question 5*, it is necessary to setup a model which will reproduce this optimization problem. At it stands this design problem include one set of design variables X_0. The optimization of this design variable is captured by the overall process proposed in this thesis. In other words, the architecting team will play the role of this overall optimizer searching for the best architecture definition. But in order to carry this optimization task, the architecting team needs to have an analysis corresponding to the architecture alternatives it wants to consider. This analysis is captured by the equality and inequality constraints presented above.

The first type of constraint $(R \le C(X_0, Att_\bullet, Spe))$ is captured by the consideration of functional relationships. These constraints are implemented by implementing the

functional flow (as presented in the previous section). These flows allows identifying the $Spe_{\bullet}$ and $R_{\bullet}$ (subsystem-level requirements and operating specifications). Using the subsystem sizing model allows us to relate subsystem-level requirements and operating specifications to the attributes of the subsystems. This relationship is represented by the constraint $Att_{\bullet} = f(\Gamma_{\bullet}, R_{\bullet}, Spe_{\bullet})$. Given an additional analysis capable of synthesizing subsystem-level attributes into aircraft-level attributes in a generic fashion, we can assume that the second type of constraints is also captured $(Att = f(X_0, Att_{\bullet}, Spe))$.

Therefore all constraints can be captured with the exception of the constraint characterizing $\Gamma_{\bullet}^{*}$. Previously we have shown that subsystem sizing could be performed based on either of the following optimization format presented by equation (16) and (17) (reproduced below).

$$Att_i = f(R_i, Spe_i) \quad \textbf{(16)}$$

$$\underset{X_i}{Max} \; V(Att_i, \Gamma_i) \quad \textbf{(17)}$$

Subject to:

- $R \le C(X_i, Spe_i)$
- $Att = f(X_i, Spe_i)$

Outcome: Optimal subsystem description: X_i^* and Att_i^*

In situation were the subsystems are sized using the functional regression-based approach, the fact that priority factors remain undetermined is acceptable. However for optimizer-based sizing, the priority factors are necessary to subsystem sizing.

4.4.2 **Coordinated Optimization or Architecture-Level Steering of Subsystem Optimization**

4.4.2.1 Optimization Process at Subsystem-Level

In order to size the subsystems in an optimal fashion, it is necessary to determine how they should be optimized in order to contribute to the maximization of the value of the architecture. In order to do so, the priority factor $\Gamma_{\bullet}^{*}$ allows for the parameterization of the subsystem objective function as a function of its attributes. For instance let us consider the following example where we want to size a turbofan to maximize aircraft range. Let us assume that the objective function of the turbofan is structured around two parameters (weight and the thrust specific fuel consumption at cruise -TSFC). These two parameters are mutually competing. Therefore for a given thrust requirement, the engine design alternatives can be represented by a Pareto front. In order to relate these subsystems attributes to the system level Figure of Merit or FoM, we shall assume that the system-level FoM is the operational range. In order to illustrate its relationship between the subsystem attribute, let us apply the Breguet range equation. For the reader's information the range equation is presented below:

$$Range = \frac{V}{TSFC}\frac{L}{D}\ln\left(\frac{W_{other} + n_{engines}W_{engines} + W_f}{W_{other} + n_{engines}W_{engines}}\right) \quad \textbf{(18)}$$

In this equation W_{other} corresponds to the total weight of the aircraft (except the engine and fuel weight). The parameter n_{engine} indicates the number of engines on the aircraft and W_f is the fuel weight onboard. V is the flight speed and L/D is the lift to drag ratio of the aircraft. If we assume that all non engine parameters are fixed, the topology of the operational range is represented in the Weight/TSFC space in Figure 97.

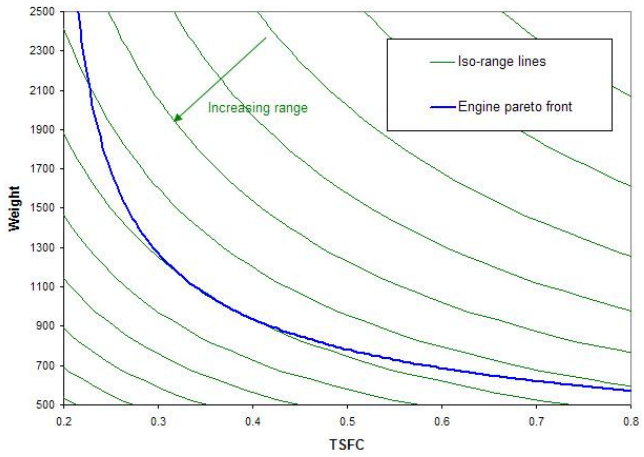


Figure 97: Subsystem front with system-level objective

The optimizer-based sizing model will detect all points on the Pareto front as valid designs. In order to determine which design to choose, the optimizer needs the objective function to specify which yields the best system performance (i.e. maximize the operational range). Since the sizing model is focused on the subsystem, it does not have access directly to the system-level FoM. In other words, the sizing model will size the model based on its immediate context of integration (i.e. the amount of thrust required from it). But it will not have access to the information necessary to perform the aircraft synthesis (i.e. the elements composing the Breguet equation), because this relationship is dependent on the architecture (number of engines, wing size and performance…). In order to substitute, this lack of system-level information on the FoM, it will receive priority factors relating the subsystem-level FoM (in this case Weight and TSFC) to the system-level FoM. For this example the objective function at the subsystem level could be defined as:

$$V_{turbofan} = \gamma_1 \times Weight + \gamma_2 \times TSFC \qquad \textbf{(19)}$$

Note: $V_{turbofan}$ is an estimation of the value of the turbofan for the architecture.

We can see that see that the form of equations (18) and (19) are very different. But I will hypothesize that if the values of γ_1 and γ_2 are proportional to the partial derivatives of the actual function relating the subsystem FoM to the system FoM, this

difference in form will not prevent the optimizer-based sizing model from identifying the subsystem solution which will optimize the system-level FoM.

Hypothesis 3.3: The architecture-level optimization solution corresponds to the location where the priority factors are proportional to the gradient of the system-level objective.

This hypothesis will be demonstrated later in this document, but we can already see visually that in order to return the real optimum, the gradient of $V_{turbofan}$ must correspond to the gradient of the system-level FoM at the real optimum. The series of plots in Figure 98 shows the solutions associated with different values of γ_1 and γ_2. We can see that changing their values will lead the optimizer-based sizing model to different subsystem solutions. The γ_1 and γ_2 set which will define the subsystem solution optimizing the system-level FoM happens to define a $V_{turbofan}$ whose gradient is pointing in the same direction as the gradient of the system-level FoM.

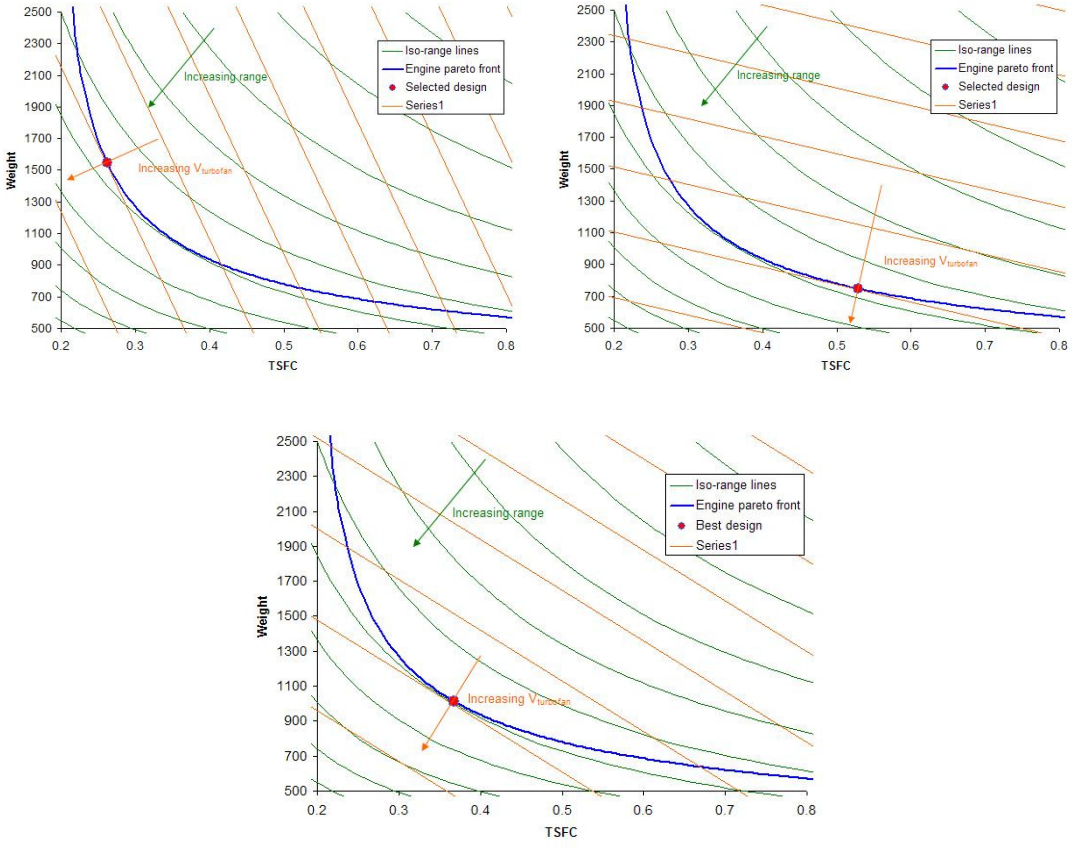


Figure 98: Observation of different optimization strategies

4.4.2.2 Optimization at System-Level

In order to identify the best subsystem solution, it is necessary to solve for the gradient of the system-level FoM with respect to the subsystem-level FoM. In the example of the Breguet range equation it could be possible to defined the partial derivative of the range with respect to engine weight and TSFC, however, this process could not be generalized to any subsystem-FoM in arbitrary architecture concepts. Therefore, solving for the partial derivative analytically is not a viable general solution for context where the integration of subsystem can be vastly changing.

In order to identify the best set of priority factors $\Gamma_{\bullet}^{*}$, an architecture-level optimizer is used. This optimizer will search for the set of priority factors that will optimize the integrated system value. This optimization process can be considered as a root search for the local derivatives of the system-level value with respect to the subsystem-level FoM. Using this image we can see the optimizer as a logical process trying out different value sets for $\Gamma_{\bullet}$. For each try, the set considered will yield a specific set of solutions for the subsystems. These subsystems solutions can be synthesized to define the system value. Based on the results provided by the $\Gamma_{\bullet}$ set the optimizer will formulate new $\Gamma_{\bullet}$ sets (i.e. new sets of priority factors) attempting to improve the overall system value.

In order implement this architecture sizing process, a multi-level optimization process is proposed: The coordinated optimization process. This process includes an architecture-level optimization process solving for ideal priority factors and a subsystem-level optimizer sizing the subsystem concept. The subsystem-level optimization process was already presented in equation (17). The Coordinated Optimization Process is presented in equation (20).

Architecture level optimizer: **(20)**

$$\underset{\Gamma_\bullet}{Max} \quad V(Att)$$

Subject to:

- $R_\bullet = f\left(Att_\bullet^*, R, X_{arch}\right)$
- $Spe_\bullet = f\left(Att_\bullet^*, R, X_{arch}\right)$
- $Att_\bullet^* = f\left(\Gamma_\bullet, R_\bullet, Spe_\bullet\right)$
- $Att = f\left(Att_\bullet^*, Spe\right)$

Subsystem sizing optimizer:

$$\underset{X_i}{Max} \quad V_i(Att_i, \Gamma_i)$$

Subject to:

- $R_i \le C_i\left(X_i, Spe_i\right)$
- $Att_i = f\left(X_i, Spe_i\right)$

X_{arc}	Parameters describing the architecture concept	R	Mission requirements (boundary function requirements)
Spe	Mission specifications	Att	Architecture attributes
$\Gamma_{\bullet}$	Priority parameters (for all subsystems)	$Spe_{\bullet}$	Functional specification dedicated to subsystems
$R_{\bullet}$	Functional requirements dedicated to subsystems	C_i	Functional capability of subsystem "i"
Definition of indexes			
$\bullet$	Indicates the collection of parameters qualifying subsystems	*	Indicates that the parameter was defined at the optimum solution of the subsystem-level optimization problem
i	Indicates the parameter is dedicated to subsystem "i"		

The proposition of this optimization implies an important hypothesis which is that coordinated optimization will yield the same solutions as a direct optimization approach (i.e. one that would directly optimize on subsystem-level design parameters).

Hypothesis 3.2: Coordinated optimization will yield the same optimized solution as a direct optimization approach.

4.4.3 **Interpretation and Implication of Coordinated Optimization**

This section has provided a description of the coordinated optimization from a sizing perspective. Via this description, we saw how solving for the "best" priority factors at architecture level allows steering the subsystem sizing processes toward the optimization of the architecture.

Now let us consider the coordinated optimization from a development strategy perspective. Earlier the following research question was formulated:

Research Question 5: How do we translate fundamental objectives into subsystem objectives?

Earlier, we hypothesized that:

Hypothesis 3.3: The architecture-level optimization solution corresponds to the location where the priority factors are proportional to the gradient of the system-level objective.

The developments of subsystem (performed in a clustered concurrent engineering context) require the formulation of design problem which will provide them both guidance and flexibility in the optimization of subsystem. This observation led us earlier to the formulation of the following research question:

Research Question 4.2: How do we translate architecture level objectives into specifications and guidelines to the subsystem developments?

Granted that *Hypothesis 3.3* is valid, the **priority factors** will provide a mean to guide subsystem developments in preliminary and detailed design phases. Using these factors allows us to recompose Taylor series expansions of the fundamental objective (system-level figure of merit) for each subsystem. The expansion will be composed only of attributes belonging to the subsystem. As a result, the priority factors will define a subsystem-level objective function for subsystem development. This objective function enables subsystem designer to perform trade-offs based on attributes that are independent of the rest of the architecture. By doing so, the coordinated optimization provides

guidelines facilitating and improving the quality of detailed design trades. But the veracity of this conclusion is subjected to the validity of the following hypothesis:

Hypothesis 3.4: Formulating the subsystem design problem as an optimization problem enables better subsystem developments.

4.4.4 **Conclusion on the Sizing Approach**

This discussion concludes the description of the methodology used to size the subsystems compositing the architecture. The previous section on subsystem knowledge preparation had introduced how the functional flows were used to facilitate the transmission of the functional requirements to subsystems. In this section the coordinated optimization scheme formulated a systematic approach to the formulation of subsystem-level objectives. With the conjunct application of these two methods the subsystem sizing problem can automatically be specified based on its architectural context. This observation now takes us back to the key part of this thesis which is the architecture concept definition.

4.5 Architecture Concept Definition

In previous sections, we have reviewed the process associated with the preparation of the architectural trade (formulation of the mission requirements, preparation of subsystem knowledge, means to optimize the subsystems). This section will now discuss a central aspect of this dissertation which is the architecture concept definition.

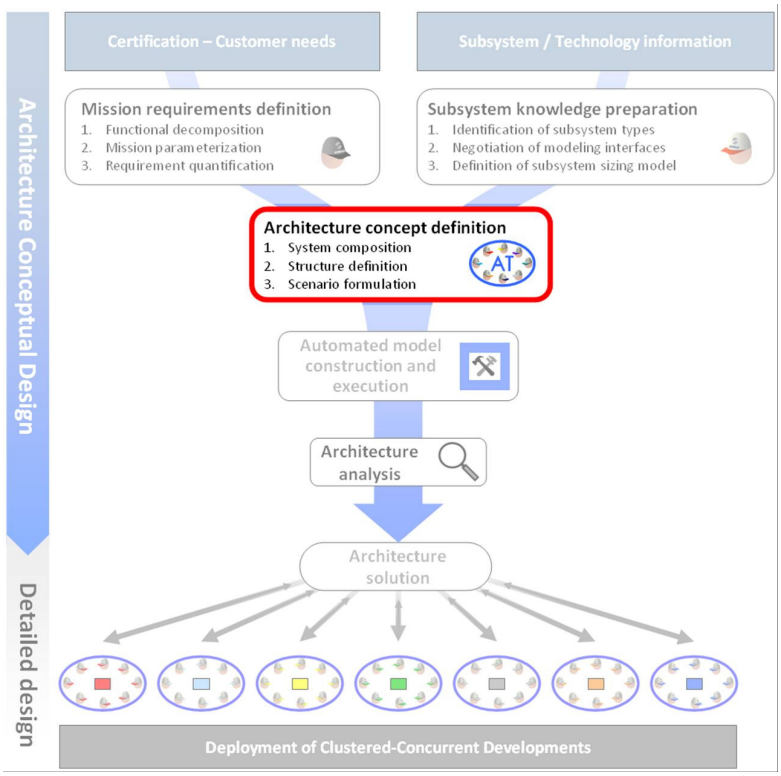


Figure 99: Process overview – Architecture concept definition

Before we introduce the methods supporting the architecture concept definition, let us consider the thought process which motivated it. Earlier in this chapter the following research question was set forward:

Research Question 2: How should we define the architecture alternatives in an unambiguous fashion while facilitating the setup of the model?

Two major aspects are implied by this research question. The first aspect concerns the information necessary to describe both an architecture concept and the model capable

of representing it. This aspect motivated the formulation of the following research question.

Research Question 2.1: Which aspects of the architecture must be captured to define its SMDA?

The second aspect of the main research question concerns the translation the architecture concept definition into the SMDA definition which can be used to size and analyze the architecture concept proposed.

Research Question 2.2: How can we translate an architecture description into an executable analysis model?

This aspect concerns the construction of the model and will be further discussed in section 4.6 (Automated Model Construction).

In this framework, we propose to construct an architecture model by assembling subsystem models. In order to know which subsystem models to integrate, we need to know which subsystem is present physically in the architecture concept. Also, earlier in this chapter, we have hypothesized (*hypothesis 2.1*) that sizing relationships between subsystem elements can be described by the functions linking them. Therefore it is necessary to define the functional relationships present in the architecture. As we will see later in this section, the functional relationships between subsystems change with the definition of the operational scenario. Therefore it is necessary to describe the architecture operationally in order to define how the architecture is performing the mission. Based on these observations, the following hypothesis is proposed to address *Research Question 2.1*:

Hypothesis 2: The functional, physical and operational description of the architecture provides an unambiguous description of the architecture and facilitates the establishment of the SMDA.

Based this hypothesis, the architecture concept definition is a process structured around three activities:

- Composition: The declaration of subsystems
- Structure: The formulation of the architecture configurations
- Operations: The definition of sizing scenarios and sequences

All these activities are performed collectively by the architecting team. A notional overview of the definition process is provided in Figure 100.

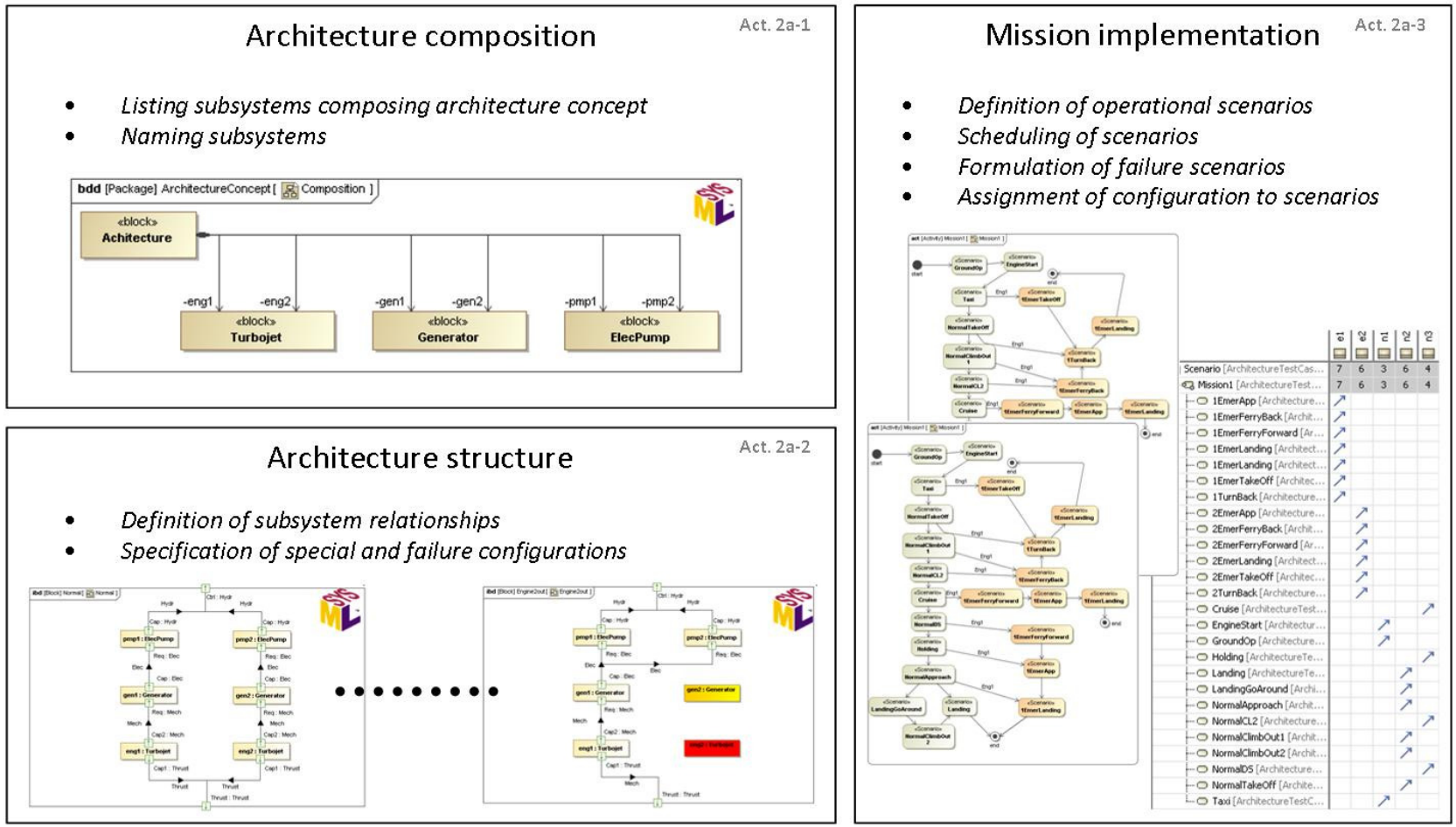


Figure 100: Overview the architecture concept definition process

4.5.1 **Composition: Declaration of Subsystems**

In order to define the composition of the architecture, we need to identify the subsystems composing it. This identification implies the following tasks:

- Identify subsystem types present in the architecture (e.g. engines, generators, etc…)
- Define the number of instances for each type (e.g. 2 engines, 4 generators, etc…) and provide a name for every subsystem instance (e.g. one engine is called "eng1", the other "eng2")

These three steps can easily be implemented using the SysML block definition diagram (BDD). The following figures provide an illustration for a composition. They will be based on an example for an architecture concept composed of two turbojets (called "eng1", the other "eng2"), two generators ("gen1" and "gen") and two pumps ("pmp1" and "pmp2").

4.5.1.1 Creation of the Architecture and Identification of Subsystem Types

In a block definition diagram, we create a new block which represents the architecture. By doing this we create an object designating the architecture. In the same diagram, we bring in the subsystem types present in the architecture by dragging them in from the library. We then link the blocks representing the subsystem types to the architecture block using a containment relationship. This relationship indicates that the block at the diamond end contains "parts" described by the block at the arrow end. At this point the diagram corresponding to the example architecture is presented in Figure 101. As it stands, this diagram specifies that the architecture is composed of a turbojet, a generator and an electric pump.

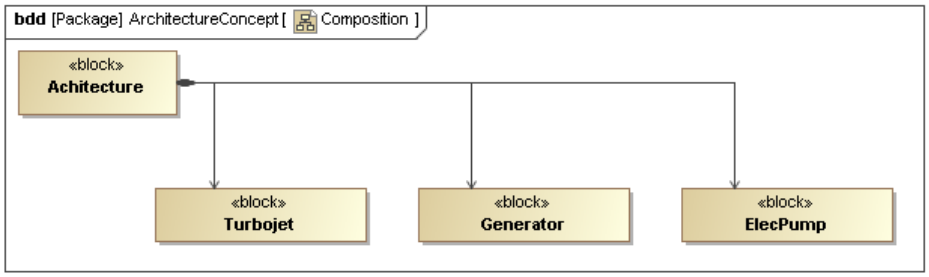


Figure 101: Step 1 in defining the architecture composition

4.5.1.2 Specification of Number of Instances and Names of Subsystems

Every time we specify a containment relationship, one part is defined. Therefore, in Figure 101, only one instance of each subsystem type was defined. If we were to specify x subsystems of the same type, it is necessary to make x containment relationships. In order to name each subsystem instance we can specify the arrow end of the relationship by assigning the name of the subsystem. To illustrate this method, the final composition diagram for the example architecture concept is presented in Figure 102.

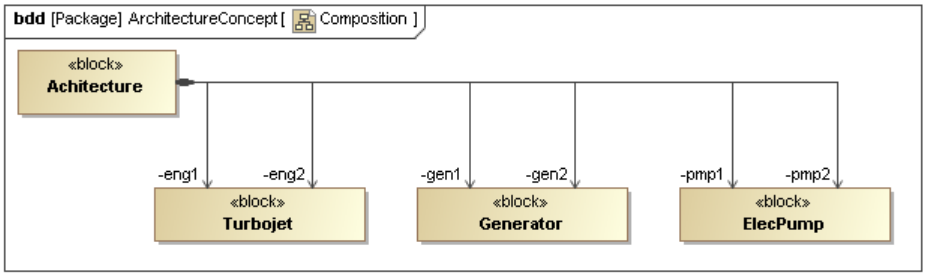


Figure 102: Step 2 in defining the architecture composition

4.5.2 Structure: Formulation of the Architecture Configurations

The relationships between subsystems define the structure. Earlier, when *Hypothesis 2.1* was introduced, we highlighted the fact that relationships between subsystems could be defined by the functions that they perform for each other. Based on this hypothesis we need to define the structure by specifying the functional architecture.

The functional architecture is defined by specifying the functional relationships between subsystems.

The challenge associated with the definition of the structure is that the functional relationships can change. For instance, the structure may be reconfigured in a situation where a subsystem has failed. To illustrate this situation let us consider the architecture concept defined in Figure 102. In normal operation, eng1 powers gen1 which itself provides power to pmp1. The others (eng2, gen2 and pmp2) are related in a similar fashion. If eng2 is failed, then gen1 provides for both pmp1 and pmp2. Therefore in some situations, gen1 is connected to pmp1 only, on others it is connected to both pmp1 and pmp2. In a similar fashion, the relationships between subsystems may change, depending on the flight phase or conditions, in order to optimize performance. For instance, most commercial aircraft can either produce their energy from their engines or from their auxiliary power unit as they stand on the ground.

In order to organize these relationships, the structure will be described by "**configuration**". A configuration is the ensemble of relationships that are in place at a given point in time. In the example, two configurations were described. The first configuration corresponds to normal operations, and the second to an engine failure.

Therefore the architecture structure is defined by an ensemble of architecture configurations. Each configuration is specified by listing the functional relationships between subsystems. This implies that the definition of the architecture process must occur at two levels:

- First we must list the configurations composing the structure
- Second we must specify the relationships composing each configuration

4.5.2.1 Listing Architecture Configurations

To instantiate architecture configurations we use a BDD. Each configuration will be defined by a block. In this BDD, we represent the architecture block (defined

previously, when specifying the architecture composition). Each new configuration must be linked to the architecture compositions block by a specialization relationship. This relationship is defined using a white headed arrow. In the example, two configurations were evoked in the description: the first configuration for normal operation and the second for eng2 failures. The following diagram presents the definition of these configurations.

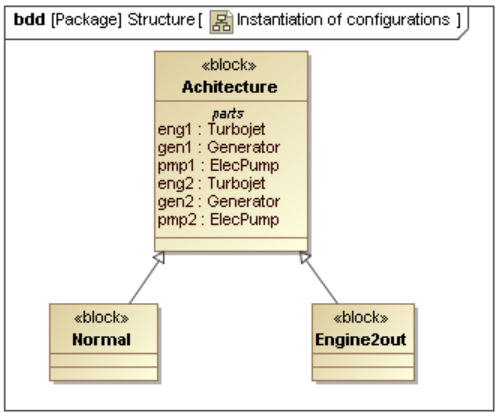


Figure 103: Introducing configurations

By defining configurations as specializations of the composition blocks, the configurations are implicitly defined as making use of the subsystem parts defined previously in Figure 102.

4.5.2.2 Definition of Architecture

The architecture configurations are defined by the ensemble of the relationships between subsystems. These can be defined using the SysML internal block diagram (IBD). The IBD specifies the relationships between the parts (i.e. subsystems) composing the block (i.e. system). The parts in the configuration blocks correspond to the subsystems (thanks to the graph presented in Figure 103).

Previously (in activity 1b-1), each subsystem type was defined functionally. The functional flows produced and required by each subsystem were defined as SysML item flow ports (illustrated as the green square with an arrow). Thanks to this definition, each

subsystem identified previously will be shown with the flow ports associated with their type. In order to define a functional relationship, an item flow must be created. An item flow is represented by a connector conveying an "item". The connector is a link between two elements (in this case the elements are the flow ports representing subsystems functional interfaces). This link represents an exchange. The "item" conveyed by the connection characterizes the exchange between the two related elements. In this case the item conveyed is the functional flow type which was defined in activity (1b-3). Using the item flow enables two important features. Its first purpose is to define the functional relationship. This definition contributes to the formulation of the configuration. Its second purpose is to identify the flow of variables necessary to characterize the functional relationship. In order to illustrate this approach, an IDB characterizing the "normal" configuration of the example architecture is provided in Figure 104. Note: the two ports located on the boundary of the graph represent the boundary functions.

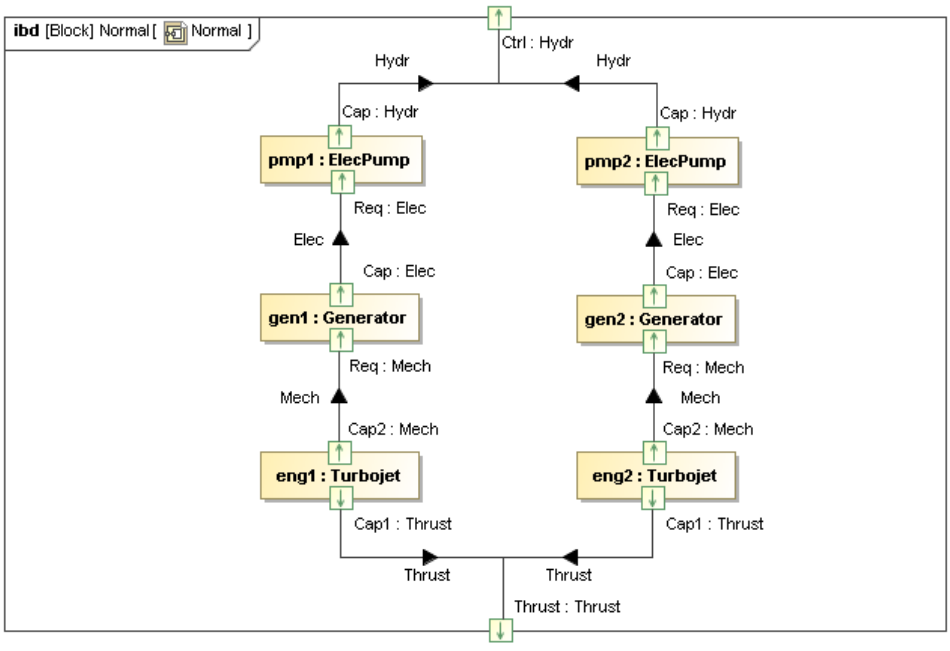


Figure 104: Definition of configuration "Normal"

The failed engine configuration would be defined in a similar fashion except that the eng2, which is expected to be in a failed state, would not be connected. If we assume

gen2 was physically mounted on the eng2, then if eng2 is failed, gen2 is also inoperative and therefore its representation in the IBD would not be connected either. The representation of the failed configuration is presented in Figure 105.

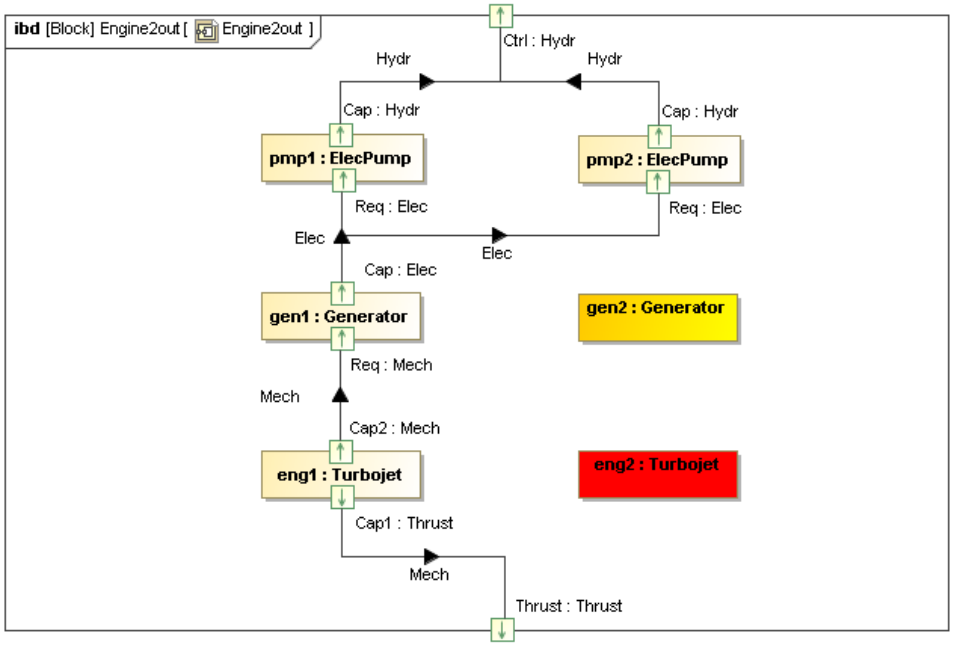


Figure 105: Definition of configuration "Engine2out"

4.5.3 **Operation: Definition of Sizing Scenarios and Sequences**

Now that we have defined how to identify the subsystems composing the architecture and how they are interconnected, we can define their scenarios of operation. The definition of the mission is defined via:

- The definition and scheduling scenarios
- The characterization of functional requirements
- Assignment of architecture configurations

4.5.3.1 Definition and Scheduling Scenarios

The first objective of the definition of scenarios is to identify the list of possible scenarios. The scheduling of scenarios is the definition of the order of execution between the scenarios. These definitions are done using the SysML activity diagram (ACT). This

diagram is a graph composed of nodes and links. Each node corresponds to a scenario. The links define the possible transitions between scenarios. The following graph provides an illustration for scenario sequences.

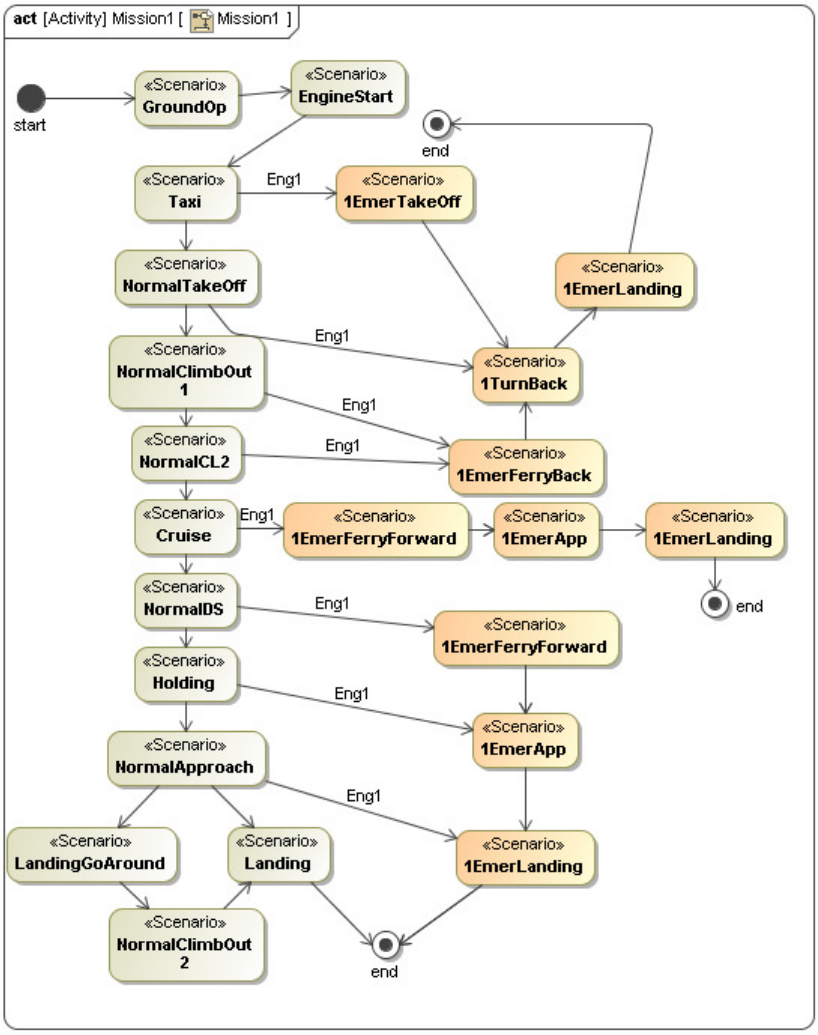


Figure 106: Definition of mission sequences

The special nodes labeled “start” and “end” are respectively initial and final nodes. They indicate the scenarios which initiate and terminate a **mission sequence**. A mission sequence is defined as a series of consecutive **operational scenarios**. The mission may be composed of multiple mission sequences.

In this thesis the usage of the activity diagram is not canonical. In this diagram we are less trying to capture the logic of transitions between scenarios than trying to list all

the possible sequences. Therefore, in order to keep the diagram as simple as possible some minor liberty was taken with respect to the semantic defined by the SysML community. Normally, the activity nodes may have only one outbound connection. The reason is that this diagram is designed to provide an unambiguous description of the sequences of activities. If a node has more than one outbound transition there is an ambiguity with respect to which transition should be considered. Therefore, if we wanted to be strictly rigorous with respect to the SysML, one should place a decision node between every activity. Since in this diagram we attempt to capture the ensemble of the possible transitions, placing these decision rules would only add elements which would serve no purpose in our application. Therefore for simplification sake, the spanning transitions are kept and their meaning specified by assigning a name to the transitions corresponding to failure events. The name assigned to the links corresponds to the name of the subsystem which failed. Transitions without a name correspond to a normal continuation of the mission (no additional failures).

4.5.3.2 Characterization of Functional Requirements

In the architecture requirement definition activity, a SysML stereotype was created to characterize scenarios. The stereotype was derived from the SysML activity and was specialized with the addition of three tags which allows the user to specify the flight phase, the flight conditions and the criticality of the configuration.

For each node defined previously in the activity diagram, it is necessary to assign them to the "scenario" stereotype and specify their tags. Specifying their tags implies choosing the relevant flight phase and condition corresponding to the scenarios along with the appropriate criticality classification for the failure configuration. Based on this definition, it is possible to define the functional requirements associated with the scenario (*Hypothesis 1: Functional requirements can be characterized by defining scenarios based on flight phases, flight conditions and criticality depths*).

4.5.3.3 Assigning Architecture Configurations to Scenarios

In the structure definition section, we saw that the architecture may operate in several configurations. This diversity in configurations results either from subsystem failure or from the need to adapt the architecture to optimize its performance. In each scenario defined previously, the boundary functions are served by a specific configuration defined via activity 2b-2 (formulation of the architecting configurations). Therefore, it is necessary to map a configuration to each scenario.

The mapping of scenarios to architecture configurations is identified by allocations of scenarios to configurations. The term "allocation" may take different meanings in the SysML. In this context, scenarios correspond to a set of requirements which are imposed (i.e. "allocated") to a specific configuration. In other words, the scenarios specify constraints which will be imposed on the sizing of subsystems via the functional relationships implied by the configuration.

The definition of allocations is performed using an allocation matrix. At the time this thesis was written, the allocation matrix was not integrated to the SysML. The matrix provides a visual tool to make allocations. The cells in the allocation matrix receive tokens which identify an allocation of the element on the row (here scenario) to the element on the column (configuration).

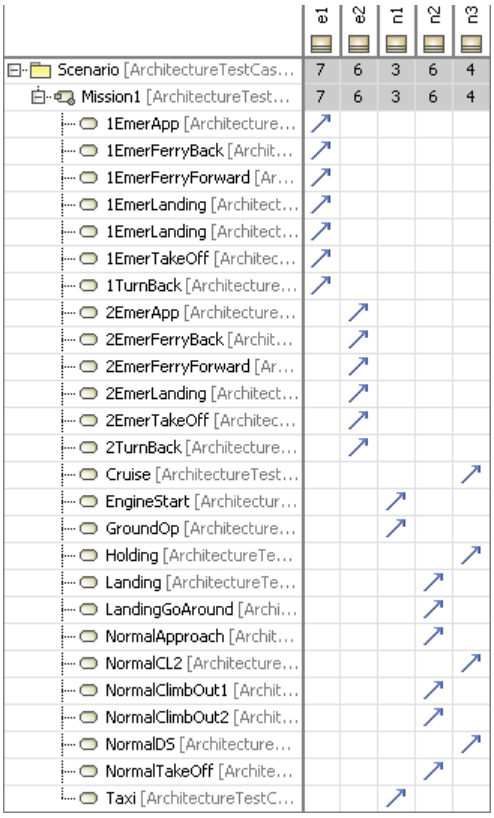


Figure 107: Allocation Matrix for scheduling configurations

4.5.4 **Conclusions**

This section has presented a SysML based process to define architecture concepts. This approach organized around the definition of the composition, structure an operation of the architecture, provides a clear breakdown of the tasks necessary to the definition of an architecture concept. The SysML diagrams provide this process with a visual, unambiguous and industry tested solution to capture this information. Besides having captured unambiguously the architecture concept, this definition is going to allow for the automated construction of the analysis model. The following section will describe this construction process.

4.6 Automated Model Construction

This section describes a method for constructing the analysis model which will represent the architecture concepts defined by the architecting team. In the subsystem knowledge preparation step the experts have prepared standardized modelling bricks capable of sizing their subsystems. The architecture concept defined subsequently provides a blueprint which indicates how the modelling bricks should be integrated to represent the architecture alternative under consideration.

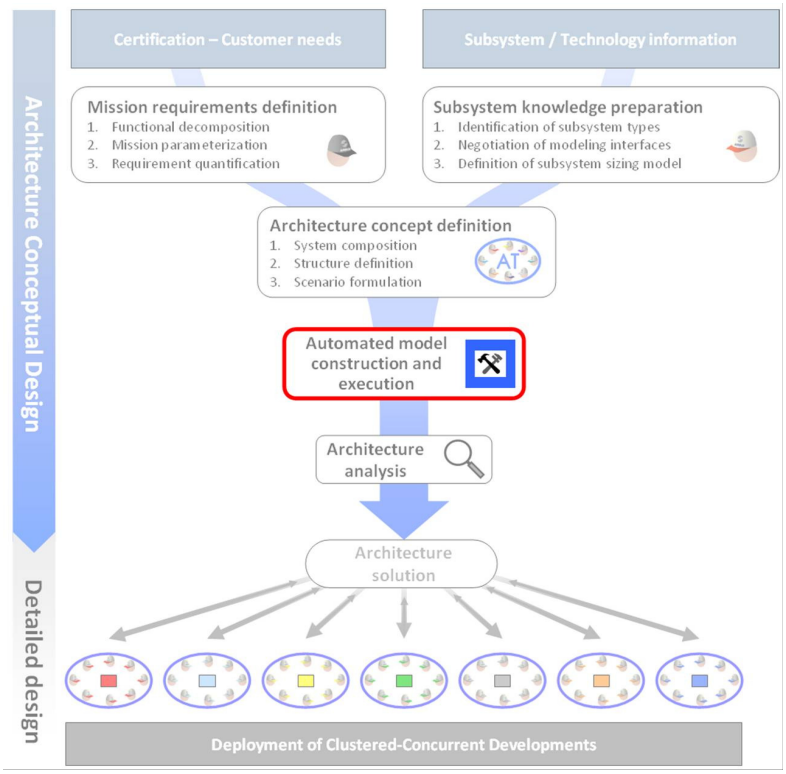


Figure 108: Process overview – Automated model construction

On this subject, the following research question was previously formulated.

Research Question 2.2: How do we translate an architecture description into an executable SMDA?

In order to address this research question, the sizing analysis is defined by considering the architecture from two perspectives: composition and Input/Output (I/O) structure.

In order to capture the specificity of each architecture concept, the model (sizing MDA) must include the sizing models corresponding to the subsystems composing the architecture under consideration. Therefore, the model composition is based on the architecture composition. This observation leads us to the formulation of a new research question refining *Research Question 2.2*:

Research Question 2.2.1: How do we translate the architecture composition into a sizing MDA composition?

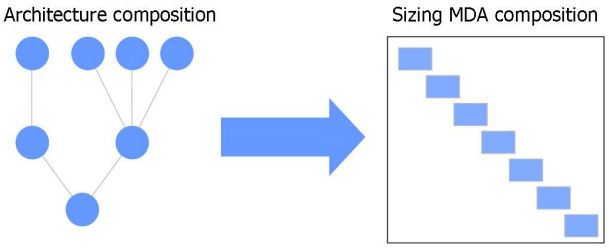


Figure 109: SMDA composition

The sizing of each subsystem is based on its relationships. Hence, these relationships define the I/O structure of the SMDA. Given the fact that the relationships are themselves defined by the structure of the architecture, the I/O structure of the SMDA is related to the structure of the architecture.

Research Question 2.2.2:: How do we translate the architecture structure into a sizing MDA I/O structure?

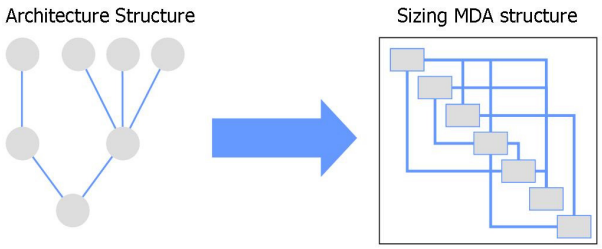


Figure 110: SMDA structure

This section will provide answers to these research questions. Up to this point we have observed already that SysML enabled us to:

- Classify subsystem knowledge (as presented in section 4.3)

- Describe the architecture (as presented in the previous section)

This section will now describe the process necessary to translate the SysML model (i.e. description of the architecture concept) into an executable numerical model which will size and analyze the considered architecture concept defined previously.

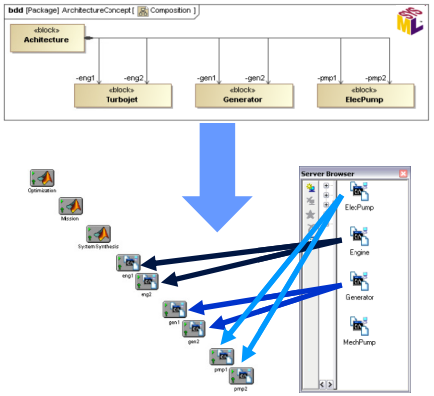
The construction process is organized in three phases.

- Phase 1: Importing the composition
- Phase 2: Framing the mission
- Phase 3: Setting up internal connections

An overview of these three phases is provided in Figure 111. The first phase of the model construction will address *Research Questions 2.2.1* by importing the models necessary to represent each subsystem. Together the second and third phases will address *Research Questions 2.2.2* by defining how the subsystem sizing model should be interconnected.

Importing composition

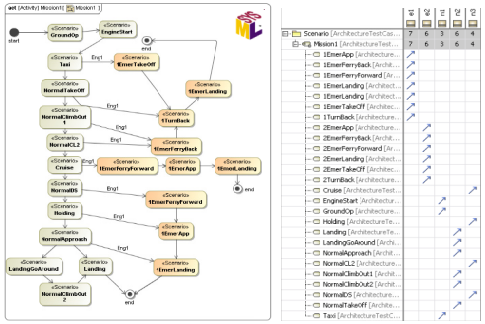
Phase 1



- *Imports subsystems modeling bricks*
- *Makes the connections necessary to*
  - *The system synthesis*
  - *The coordination of subsystem optimization*

Framing the mission

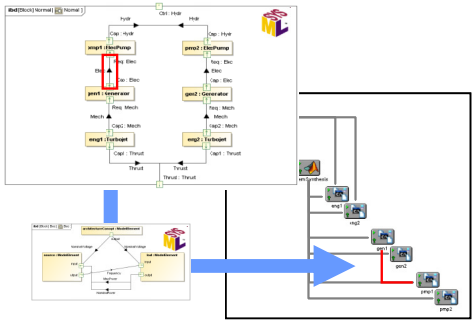
Phase 2



- *Imports*
  - *Scenario definition*
  - *Transitions between scenarios*
  - *Configurations assigned to scenarios*
- *Constructs all possible sequences*
- *Defines states series for subsystem sizing*

Setup internal connections

Phase 3



- *Construct connections between subsystem models*
- *Define connections based on functional flows*
- *Iterative process over:*
  - *operational states*
  - *Active functional relationships in the state*

Figure 111: Overview of the model construction process

4.6.1 **Phase 1: Importing the Architecture Composition**

4.6.1.1 Description

Phase 1 defines the composition of the model based on the composition of the architecture. It does so by reading the SysML graph in order to detect the subsystems composing the architecture. Each subsystem is then represented by its sizing model. Once imported in the model, some basic connections are setup. These connections will allow for:

- The integration of the optimization levels (i.e. relate subsystem-level processes to the architecture-level optimization)
- The system synthesis (i.e. integrate subsystem attributes to determine the architecture level performance)

This approach is based on the following hypothesis addressing *Research Question 2.2.1*:

Hypothesis 2.2: The composition of the model corresponds to the composition of the architecture.

4.6.1.2 Example

Each part in the architecture block corresponds to a subsystem. For each subsystem, a sizing model is created. The type of the model is prescribed by the type of subsystem.

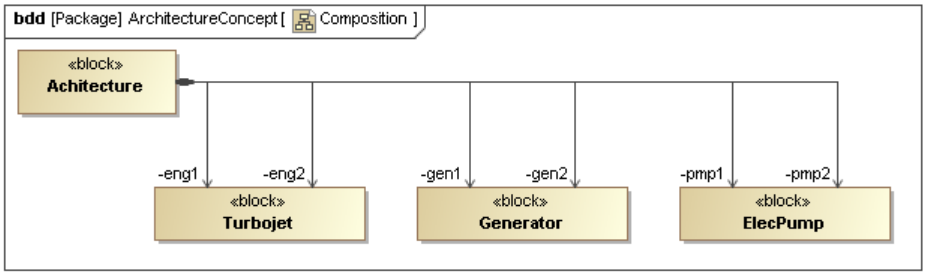


Figure 102 (reproduced): Example of an architecture composition

Figure 102 presents an architecture composed of 6 subsystems. Therefore, the builder will create 6 model elements. Two of these model elements will be instances of the turbojet sizing model, another two will be generator sizing models, and the final two will correspond to electric pump sizing models. To illustrate this process a snapshot of a Model Center model representing the architecture is shown (see Figure 112).

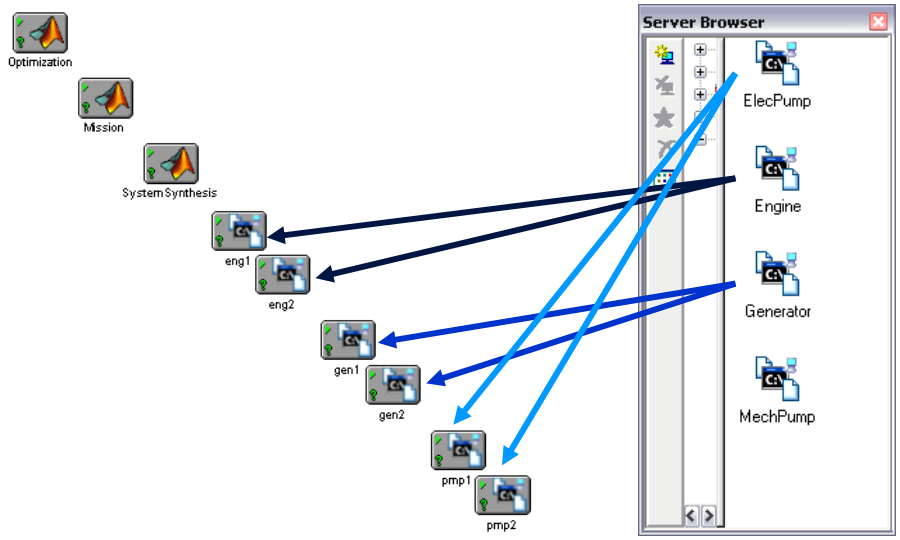


Figure 112: Importing the architecture composition (phase 1 - step 1)

For the subsystems subjected to optimization (models based on the optimizer based sizing approach), it is necessary to reconnect the sizing optimization process to the architecture level optimizer. These connections are defined as part of phase 1. In this example, let us assume that the engines are subject to an optimizer-based sizing process. Therefore, their sizing requires that they receive priority variables from the architecture-level optimizer (hypothesis 3.2). This set of connections is represented in Figure 113.

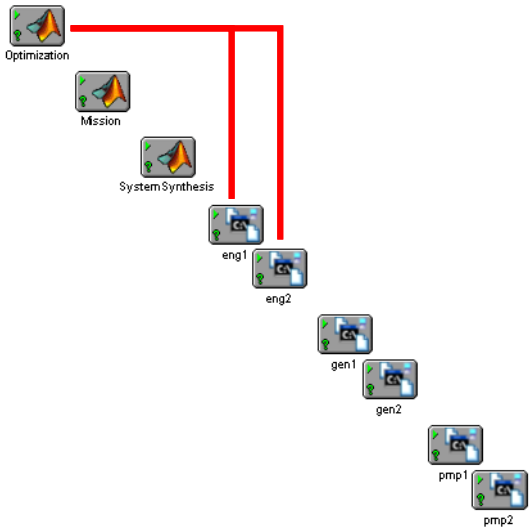


Figure 113 : Setting up the optimization connections (phase 1 –step 2)

In order to be able to perform the synthesis of the architecture, it is necessary to transmit the information about the subsystem attributes to the system synthesis analysis. This analysis (third box from the top in Figure 113) has a variable defined to receive the attributes of all potential subsystems in the architecture. The connection necessary to the transmission of this information is set as part of phase 1. To illustrate the establishment of attribute connections, let us consider the mass of subsystems. This attribute is defined subsystem by subsystem by their respective sizing model. The total must be determined at the architecture-level in order to define the architecture total weight.

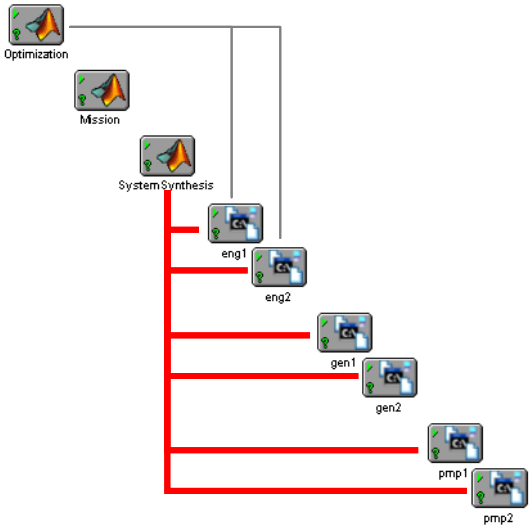


Figure 114: Returning subsystem attributes to system synthesis (phase 1 – step 3)

Therefore, as subsystem models are imported into the architecture model, connections are created in order to return their key attributes to the system synthesis analysis. These connections are highlighted in Figure 114.

4.6.2 **Phase 2: Importing the Mission**

4.6.2.1 Description

This phase is organized around four steps. In the first two, the builder reads-in the activity diagram representing the possible **mission sequences**. In step 1, the diagram is used to scan the definition of each scenario. For each scenario it will access and record its definition in terms of mission parameters (flight phase, condition and criticality) and allocated configuration. In step 2, the tree is read for its transitions. Using the transitions, the builder will detect all possible sequences of scenarios defined in the activity diagram (step 3). To do so, it will detect all possible paths from the initial node to the final node. In step 4, the sequences defined previously are used to define all the possible states in which the architecture may have to operate. In this context, an operational **state** corresponds to a scenario within a given sequence.

4.6.2.2 Example

To illustrate the actions performed in phase 2, let us consider an example based on the activity diagram presented in Figure 115.

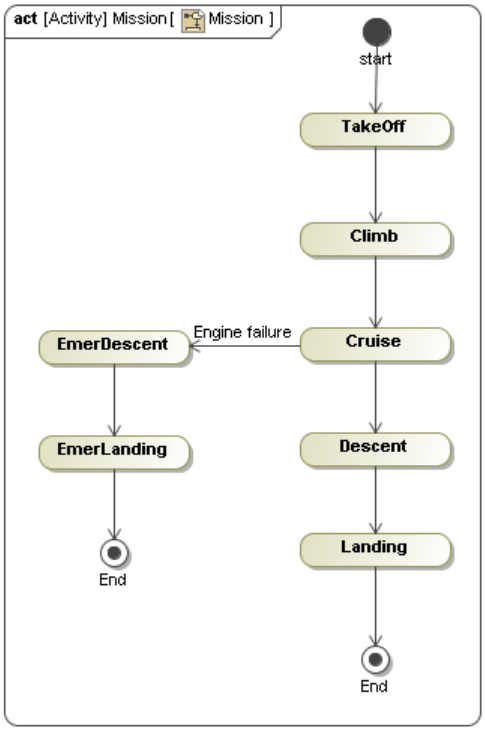


Figure 115: Simple mission sequence

In step 1, the builder reads-in the tree and extracts the information stored in the definition of the seven scenarios present in the graph. To illustrate this process, the specification of the scenarios detected in the tree above is presented in Table 14

Table 14: Specification of operating scenarios (phase 2 – step 1)

Scenario	Flight phase	Flight conditions	Criticality level	Configuration
TakeOff	1	ISA	No failure	Normal
Climb	2	ISA	No failure	Normal
Cruise	3	ISA	No failure	Normal
Descent	4	ISA	No failure	Normal
Landing	5	ISA	No failure	Normal
EmerDescent	4	ISA	Emergency	Engine2out
EmerLanding	5	ISA	Emergency	Engine2out

From the tree presented in Figure 115, the builder would then read in the transitions between the scenarios. This information is then stored in a matrix (reproduced in Table 15)

Table 15: Importing scenario transitions (phase 2 - step 2)

	TakeOff	Climb	Cruise	Descent	Landing	EmerDescent	EmerLanding	End
Start	↑							
TakeOff		↑						
Climb			↑					
Cruise				↑		↑		
Descent					↑			
Landing								↑
EmerDescent							↑	
EmerLanding								↑

Based on the information stored in Table 15, the builder scans all possible paths to go from the start node to the end node. In this example, two sequences would be detected in step 3 (see Table 16).

Table 16: Detection of possible mission sequence (phase 2 – step 3)

Sequence 1	TakeOff	Climb	Cruise	Descent	Landing
Sequence 2	TakeOff	Climb	Cruise	EmerDescent	EmerLanding

Both the first and second sequences are composed of 5 scenarios. Based on these two sequences 11 states would be defined in step 4. The first 5 states would correspond to the scenarios of the normal flight sequence. The sixth state remains empty to specify that the next state corresponds to a new sequence. The last five states then correspond to the engine failure sequence.

Table 17: Definition of mission states (phase 2 – step 4)

State rank	Scenario	Flight phase	Flight conditions	Criticality level	Configuration
1	TakeOff	1	ISA	No failure	Normal
2	Climb	2	ISA	No failure	Normal
3	Cruise	3	ISA	No failure	Normal
4	Descent	4	ISA	No failure	Normal
5	Landing	5	ISA	No failure	Normal
6	Blank	NA	NA	NA	NA
7	TakeOff	1	ISA	No failure	Normal
8	Climb	2	ISA	No failure	Normal
9	Cruise	3	ISA	No failure	Normal
10	EmerDescent	4	ISA	Emergency	Engine2out
11	EmerLanding	5	ISA	Emergency	Engine2out

4.6.3 **Phase 5: Building the Structure**

Previously the following research question was formulated:

Research Question 2.2.2: How do we translate the architecture structure into a sizing MDA I/O structure?

In this phase, the objective of the builder is to bring the correct functional requirements and specifications to each subsystem model in the architecture. This is done by creating connections, between the input/outputs of the subsystem models. Previously we saw that the architecture structure is defined by functional relationships (*Hypothesis 2.1*). We also saw that depending on the operational scenario, the structure can take different forms (configurations). In phase 2, the architecture builder defined a table of states (Table 17). For each state, a configuration is identified. Based on this observation we can formulate the following assertion:

Assertion: The operations allow for the determination of when functions occur.

Hypothesis 2.1 (reproduced): The functional relationships between two subsystems characterize the flow of information between their sizing processes.

If we associate this assertion with *hypothesis 2.1*, we can see that both the operational and functional descriptions are necessary to the definition of the flow of variables between subsystem models. Since the I/O structure of the architecture sizing model is defined by the ensemble of the variable flows, we can say that *Research Question 2.2.2* is addressed by the conjunction of *Hypothesis 2.1* with the assertion above.

Also the operational description (imported in phase 2), provides a description of flight phase, condition and criticality. Based on *Hypothesis 1*, this definition allows for the definition of boundary requirements (functional profile) and assignment of these requirements to physical subsystems (via the relationships implied by the configuration). Therefore, the construction of the I/O structure of the model is in fact the convergence point of several hypotheses set forward in this work.

4.6.3.1 Description

In order to size each subsystem properly, it is necessary to observe the ensemble of operating conditions. This requires the definition of the functional requirements for each operating state implied by the mission. Therefore the builder will construct the connections state by state. For each state the builder:

- Accesses the architecture configuration used in the scenario (step 0)
- Lists the functional relationships composing the configuration (step 1)
- Identifies the flow of variables implied by the function for each relationship (accessing the functional flow) (step 2)
- Defines the formula for each connection implied by the functional flow. (step3)

An overview of this process is provided in Figure 116.

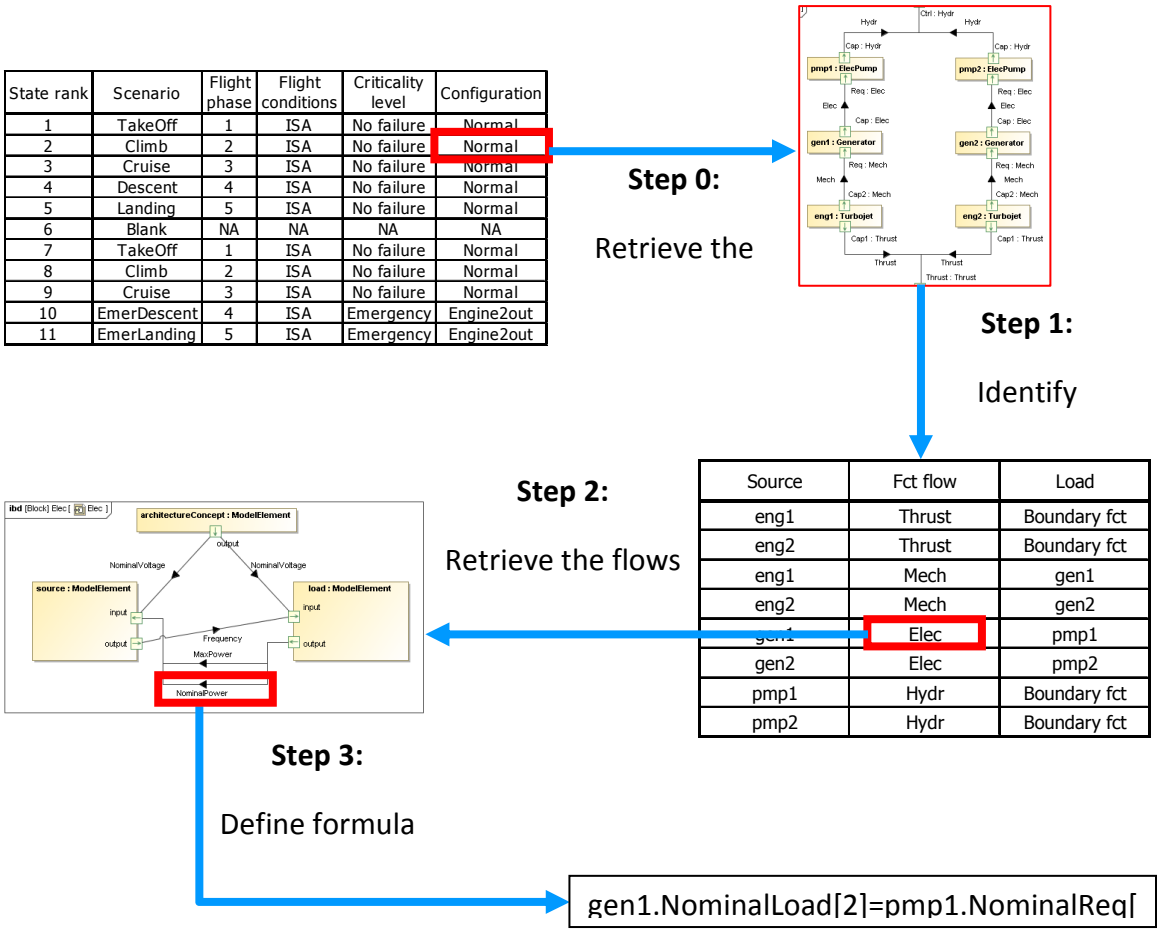


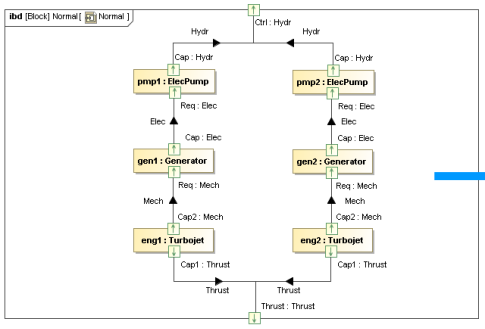
Figure 116: Structure builder process (phase 3)

4.6.3.2 Example

In Table 17, eleven states were defined. The structure process is an iterative process which is applied to each state. For each state, the architecture configuration is retrieved (step 0). The IBD defining the configuration is read by the builder which identifies all connections (step 1). The connections are defined by three main elements which are:

- the subsystem providing the function (the source)
- the subsystem receiving the function (the load)
- the type of function relating the two elements.

The following table provides an example of translating an IBD into a table listing the functional relationships.



Source	Fct flow	Load
eng1	Thrust	Boundary fct
eng2	Thrust	Boundary fct
eng1	Mech	gen1
eng2	Mech	gen2
gen1	Elec	pmp1
gen2	Elec	pmp2
pmp1	Hydr	Boundary fct
pmp2	Hydr	Boundary fct

Figure 117: Identification of functional relationships (phase 3 – step 1)

The following step iterates on each of the connections composing this configuration. In this example, there are 8 connections hence it will be performed 8 times. For each connection, the builder accesses the IBD defining the functional flow corresponding to the function tagged on the connection. This IBD (defined in Activity 1b-2 – definition of functional flows) specifies the exchange of variables implied by the functional relationship.

Source	Fct flow	Load
eng1	Thrust	Boundary fct
eng2	Thrust	Boundary fct
eng1	Mech	gen1
eng2	Mech	gen2
gen1	Elec	pmp1
gen2	Elec	pmp2
pmp1	Hydr	Boundary fct
pmp2	Hydr	Boundary fct

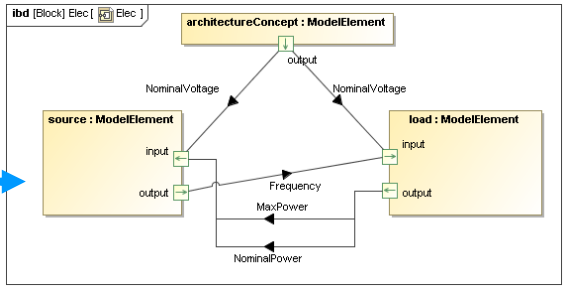


Figure 118: Retrieving the flows of variables (Phase 3 – step 2)

Step 3 is performed iteratively on each flow defined in the IBD. In the example presented in Figure 118, five flows are defined. Therefore Step 3 will be performed four times. For each flow, a connection is created. This connection requires the definition of a formula which equates one variable to another. For each flow defined in this IBD, an item is assigned. This item contains a naming convention allowing the builder to retrieve the name of the variables involved in the connection. By recomposing the name of the variables, a connection can be created by defining a formula characterizing it. For instance, let us assume that the naming convention for the nominal power flow indicated in Figure 118 specifies that the input on the source side caries the name “NominalLoad” and the output on the load side caries the name “NominalReq”. Using this naming convention allows the builder to generate a formula describing the connection between the subsystem models (see example in Figure 119).

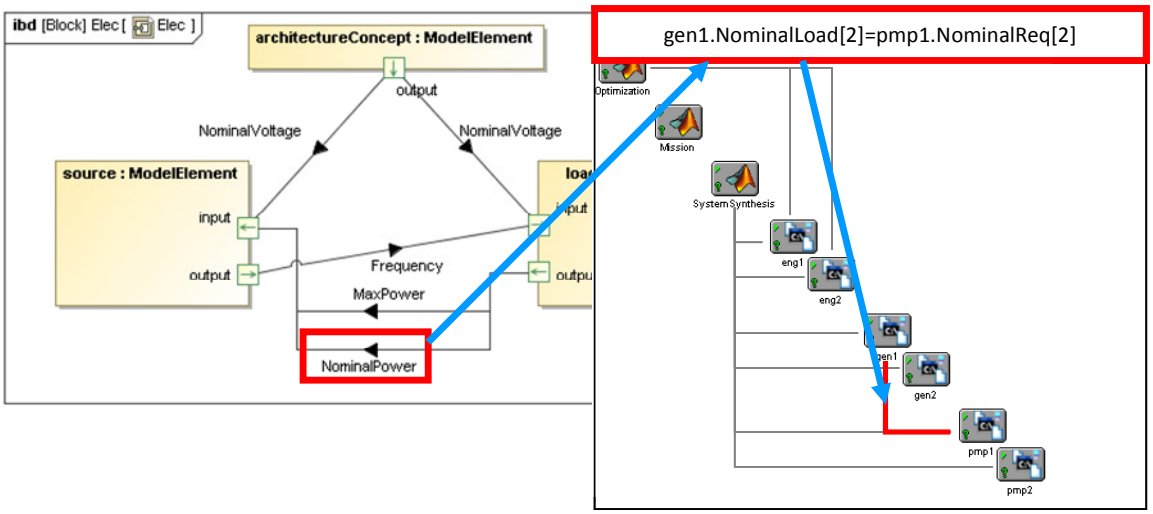


Figure 119: Definition of the connection (phase 3 - step3)

Making the links relating the subsystems may become a large iterative problem. By structuring this process, the construction of the links can be performed automatically. This automation facilitates the establishment of very large and complex models without any added work from the designer beyond defining the architecture graphically.

4.6.4 **Conclusions**

The methods presented in this section enable the automated construction of architecture models. For large architectures, the construction of such a model becomes a colossal undertaking. If performed by hand, this task would require days, weeks or possibly months. Using the methods defined this task is performed in a systematic and effort-less fashion. By facilitating the creation of new models, this approach also facilitates the exploration of architecture concepts.

4.7 Transition to Experimentation

The process presented in this chapter presented a series of methods supporting the proposed conceptual architecture design activity. The methods presented so far are based on hypotheses. For clarity, the hypotheses formulated in this chapter are reproduced below. The ideas underlying each hypothesis originated on the observations presented in previous chapter (motivation, problem statement and the review of the state of the art). But these hypotheses are also based on an inductive thought process which must now be demonstrated. This demonstration is based on a series of experiment which will be presented in the subsequent chapter.

Table 18: Review of Hypotheses

Hypothesis	Definition
1	*The functional requirements can be characterized in a complete, practical and accurate fashion by defining scenarios based on flight phases, flight conditions and criticality depths.*
2	*The functional, physical and operational description of the architecture provides an unambiguous description of the architecture and facilitates the establishment of the SMDA*
2.1	*The functional relationships between two subsystems characterize the flow of information between their sizing processes.*
2.2	*The composition of the model corresponds to the composition of the architecture.*
3.1	*The sizing process is an approximation of an optimization process.*
3.2	*The architecture-level optimization solution corresponds to the location where the priority factors are proportional to the gradient of the system-level objective.*
3.3	*Coordinated optimization will yield the same optimized solution as a direct optimization approach.*
3.4	*Formulating the subsystem design problem as an optimization problem enables better subsystem developments*

Chapter 5

Verification of Hypotheses

In the previous chapter, several hypotheses were formulated and will be verified in this chapter. Before we move on to the verification itself let us take a step back and observe the general argumentation set forth so far. The fundamental objectives and research questions are presented in the figure below.

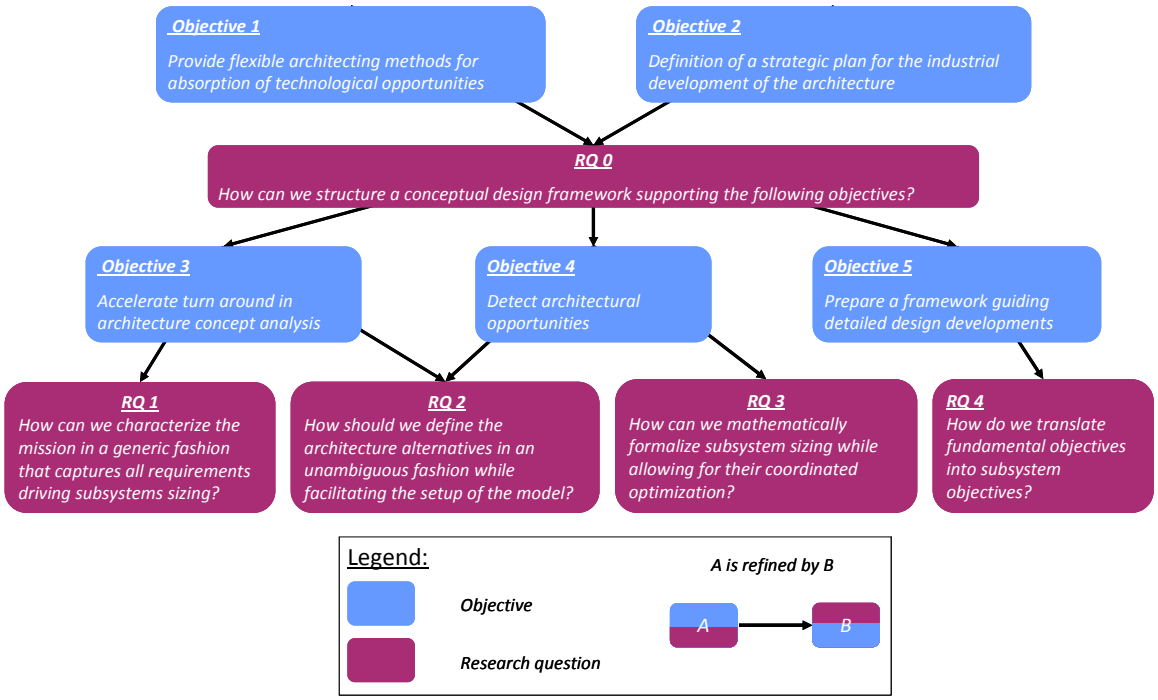


Figure 120: Overview of fundamental research questions and objectives

In Chapter 1, *Objectives 1* and *2* where formulated. The analysis of industrial practice in Chapter 2 led us to *Research Question 0* and *Objectives 3*, *4* and *5*. In order to address these fundamental objectives, an assertion was formulated which led us to the investigation of an IPT/MDA approach to architectural conceptual design. After reviewing the state of the art pertaining to the conceptual design of system architectures in Chapter 3, four general *Research Questions 1*, *2*, *3* and *4* were formulated. In order to address these research questions the methodology presented in Chapter 4 was presented.

This methodology is supported by several hypotheses. In order to validate these hypotheses, a set of four experiments will be presented in this dissertation. These experiments are of two types. The first three experiments will be focused on the validation of specific hypotheses while experiment 4 will demonstrate the application of the overall thesis on a large test case. An overview of the entire analysis process and how it relates to the following experiments is presented in Figure 121.

This present chapter will present the first three experiments. The description associated with each is presented in a dedicated section in this chapter. Experiment 4 has a larger purpose than testing a specific hypothesis as it provides a test case and illustration for the overall process. For this reason experiment 4 will be discussed and described in the following chapter.

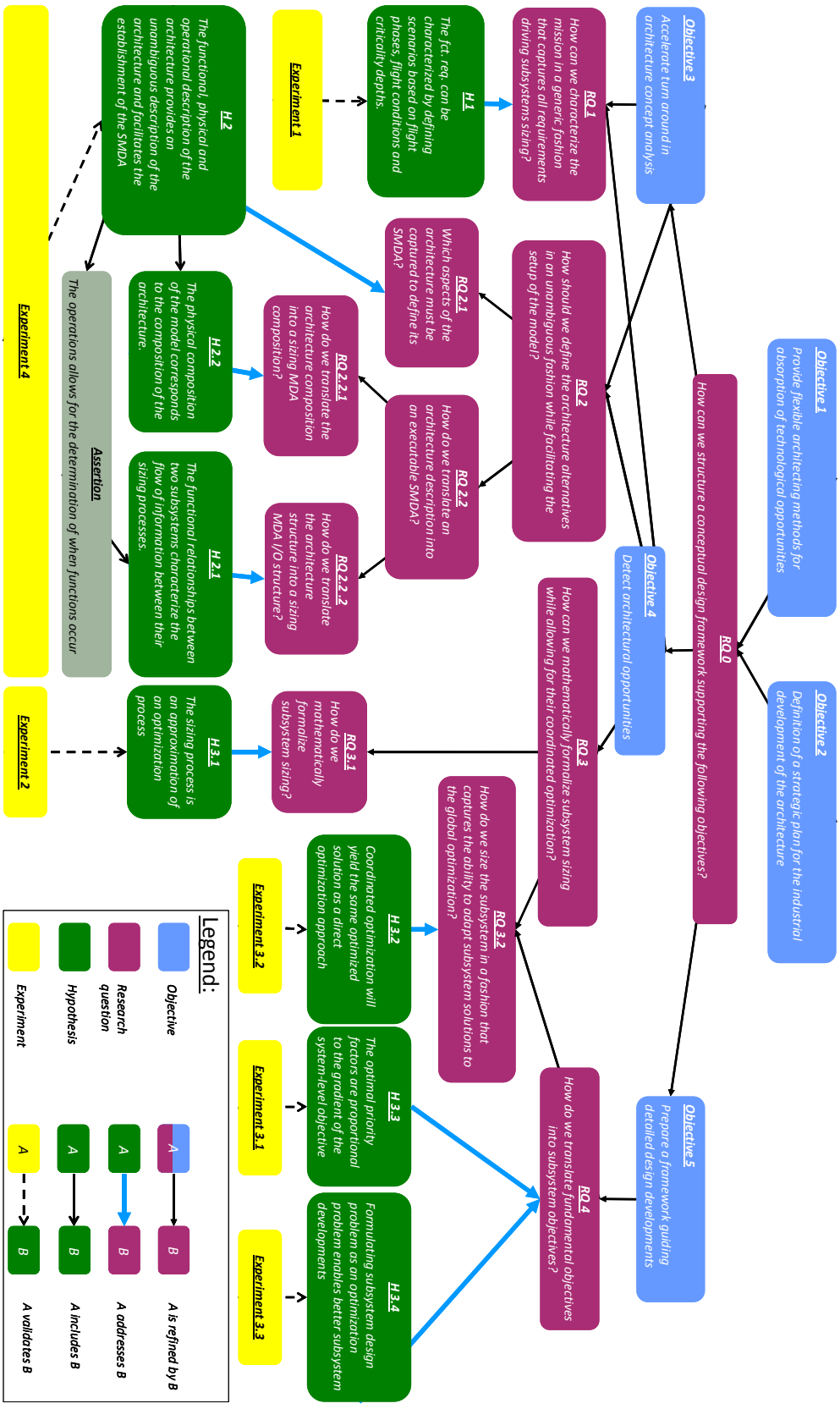


Figure 121: Overview of the philosophical process

5.1 Experiment 1: Characterization of a Commercial Aircraft Mission

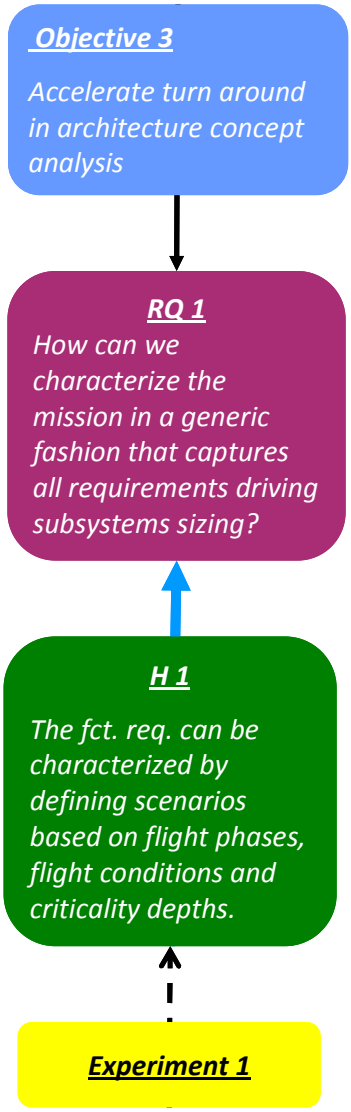


Figure 122: Experiment 1 overview

Based upon *Hypothesis 1*, a methodology was formulated to capture in a generic way the complex mission of a commercial aircraft. This methodology supports the mission requirement definition (as described in Chapter 4). In order to verify the validity of the hypothesis, the methodology will be applied to the mission of a commercial passenger transport aircraft (single-aisle class). The verification process used in this experiment will assume that the ability of the methodology to capture this mission will qualify the validity of *Hypothesis 1*.

The test case in this experiment will correspond to the mission of an A320 class aircraft. This experiment will focus on two main aspects:

- The ability to represent the very diverse requirements imposed by the mission
- The ease with which is it possible to characterize the requirements in a generic way

In order to validate the hypothesis, the methodology will have to satisfactorily perform on these two attributes.

5.1.1 Application of the Methodology

In this example, only two **mission parameters** will be used: the flight phase and the criticality level. In order to implement the tasks implied by the "mission requirements definition" task, we must define the flight phases (and what they imply in term of needs). We must also define the general profile of what is implied by the criticality levels considered and what they imply. These definitions will be followed by a description of how the various requirements are elicited.

5.1.1.1 Functional Breakdown

In order to breakdown the mission of a commercial transport aircraft, a **functional breakdown** derived from the work by the OAPA team [28] was used. This breakdown lists the **boundary functions** (i.e. functions directly defined by the mission and architecture independent). Figure 123 presents the breakdown. A description of the main functions composing it is provided in Table 19.

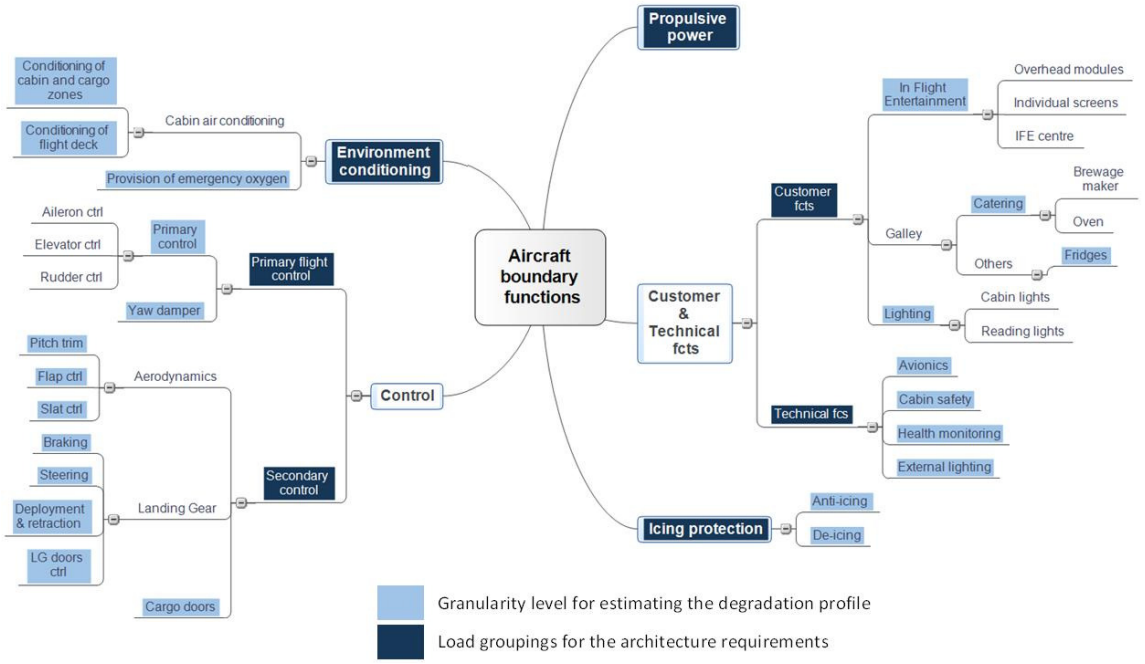


Figure 123: Functional breakdown of a commercial transport aircraft

This functional breakdown was used to define the functions imposing energy loads on the architecture. Those loads were defined at two levels:

The first level defines the load groupings (dark blue). The load groupings define the functions that will be imposed on the architecture. In other words, when the architecture will be sized, it will be subjected to loads grouped at that level of granularity. Therefore, the quantification of the functional requirements will be performed at that level.

The second level corresponds to the sub-functions (light blue). The energy consumption of smaller (and therefore simpler) functional elements can be estimated more clearly than large aggregated functions. Using this lower level itemizes the sub-functions implied by the mission. This itemization is also very useful in order to define more precisely the fluctuation of the loads during the mission or to realistically capture the load shedding. Instead of shedding (or abstaining from degrading) the entire function group, using this lower level of granularity captures more precisely the degradation (both necessary and allowable).

Table 19: Description of boundary functions

Function or energy/power load	**General description**
Thrust	The power architecture shall provide the means to propel the aircraft. This functionality is quantified by take-off/landing field length, and climb rate requirements.
Environment Conditioning System (load)	The Environment Conditioning System (ECS) shall regulate the cabin (including passenger, cargo and cockpit areas) pressure. This system is also responsible for the ventilation and temperature control of these zones.
Wing Icing Protection System (load)	In case of presence of super-cooled water during certain phases of flight, ice accretion on the wing shall be avoided. This functionality implies that electric mats are used to heat the leading edge of the wing.
Customer loads	This functionality is decomposed further into four sub-functions. This function is dedicated primarily to the comfort, entertainment and hosting activities.
Technical loads	Loads dedicated to avionics
Primary control	Functionality dedicated to the flight control of the aircraft. This includes the energy used by control surface actuators (ailerons, spoilers, rudder and elevator).
Secondary control	Loads dedicated to the actuation of mobile elements not directly dedicated to the flight control of the aircraft.

5.1.1.2 Definition of Flight Phases

The mission considered is a standard mission for a single-aisle commercial transport aircraft. An overview of this mission is presented in Figure 124. The mission has been organized in 12 **flight phases** presented in the following paragraph.

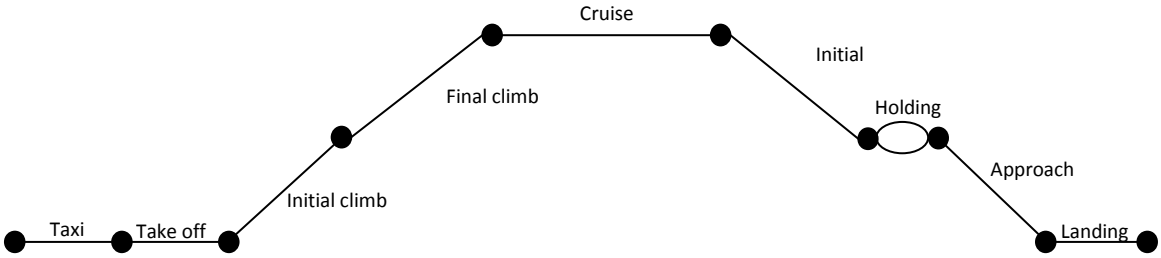


Figure 124: Mission sequence overview

This mission is presented in 11 flight normal phases plus one emergency flight phase. These phases are used to characterise the functions that will be required from the architecture. A general overview of the flight phases considered is provided in the following paragraph.

5.1.1.2.1 Introduction of Flight Phases

Figure 125: Phase 1

Parked: The aircraft is parked at the gate. The power is provided by an external source. Maintenance and testing functionalities are susceptible to be performed.

Figure 126: Phase 2

Engine start: The engine start sequence is initiated (on all engines simultaneously). Other functionalities are on stand-by mode.

Figure 127: Phase 3

Taxi to the runway: The aircraft should propel itself to the runway. The taxi speed is set at ~ 50 km/hour. The aircraft is susceptible to be exposed to icing conditions.

Figure 128: Phase 4

Take off sequence: The aircraft accelerates, rotates and lifts off. The aircraft shall have the capability to resist icing conditions. The propulsive power shall be sufficient to takeoff within specified distance.

Figure 129: Phase 5

Initial climb: The aircraft shall sustain a rate of climb sufficient be certifiable and/or to minimize its noise foot print while retracting its high-lifting surfaces and landing gears. The aircraft shall be protected from icing conditions.

Figure 130: Phase 6

Final climb to cruise altitude: The aircraft shall climb to a cruise altitude set at FL 350 with a specified climb rate. The aircraft is susceptible to be exposed to icing conditions. The aircraft shall provide enough energy for customer (cabin) functionalities.

Figure 131: Phase 7

Cruise: The aircraft should be cruising as far as fuel reserve allow. In cruise, the power architecture shall accommodate power demands from customer (cabin) functionalities.

Figure 132: Phase 8

Initial descent: This phase characterizes the beginning of the descent once the aircraft is arriving to destination. In this flight phases customer functions shall be maintained.

Figure 133: Phase 9

Holding: The aircraft is on stand-by at FL 100 waiting for clearance to land. The aircraft shall fly in this phase for 20 min.

Figure 134: Phase 10

Final approach: This phase characterizes the final descent before landing. The high-lifting surfaces and the landing gears shall be deployed, and the wing protected from icing conditions.

Figure 135: Phase 11

Landing: The aircraft must decelerate after touchdown. The max thrust capability shall allow a potential go-around manoeuvre. In this phase the aircraft is exposed to icing conditions.

Figure 136: Phase 12

Ferry to emergency landing: this phase only occurs in situations where the aircraft experiences a failure which an emergency landing. This phase represents the flight leg necessary to get to the closes airport. If the concept investigated by this study was a twin engine concept, the duration of this phase would correspond to the ETOPS (Extended-range Twin-engine Operational Performance Standards) time at which the aircraft is expected to be certified.

5.1.1.2.2 Definition of Flight Conditions Associated with each Flight Phases

The methodology implicitly defines a quasi-static representation of the mission. Therefore the analysis of each flight phase will consider the rates of energy (i.e. power) and assumes that this rate is constant or at least representative of the entire phase. In order to accommodate this assumption, several measures were taken. The first measure concerns the qualification of transfer of energy. The energy transfer between subsystems is qualified by at least two values. The first value specifies the maximum power over the phase. This first value allows for an accurate sizing of the power system. The second characterizes the required energy. This value is used to capture the sizing constraint on energy systems (batteries, fuel tanks, etc…).

The other measure concerns the sizing of the propulsive system. The sizing constraints on air breathing engines are strongly dependent on the Mach number and ambient conditions (i.e. altitude). For diverse flight phases like climb and descent, using only a single point to characterize the entire phase could limit our ability to capture the engine sizing constraints (which changes with altitude and speed). In order to palliate this problem, the flight conditions selected for the 11 flight phases composing the mission have been adjusted to unsure that most constraining power requirements are captured.

This adjustment is based on the assumption that the maximum power requirements from the take-off/climb sequence are symmetric with those in the descent/landing sequence. This assumption is based on the fact that at any point in the descent sequence, the pilot must be able to reinitiate a climb (e.g. due to changes in the flight plan, to go over the weather, or go around at landing). Using this symmetry in operation, the mission has been “folded” around the cruise phase as shown in Figure 137. This folding allows considering more flight conditions in the climb phases therefore capturing a greater diversity of thrust requirements.

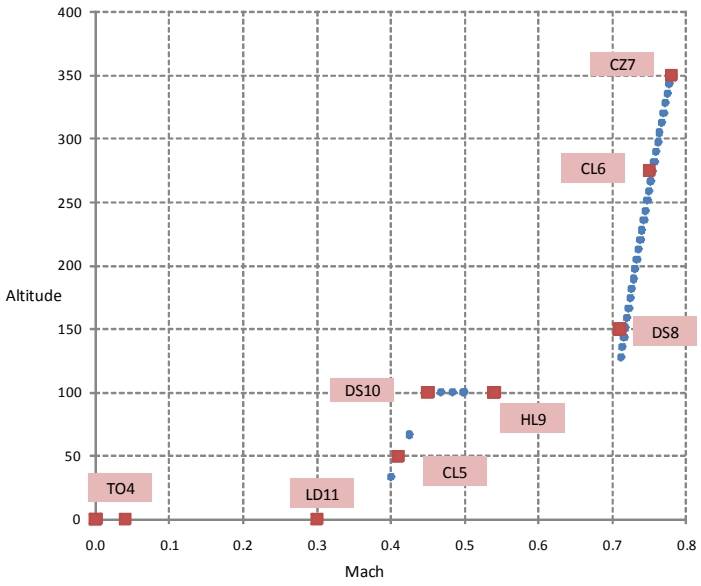


Figure 137: Distribution of flight phases

Since the engine is a power system (i.e. its sizing is driven by maximum power), only the maximum power requirements corresponding to the descent sequence were mapped to correspond to the climb. The nominal propulsive power in descent was not mapped with what they would have been in climb. Based on the consideration described above, the following flight conditions, and thrust requirements were assigned to each flight phases:

Table 20: Flight conditions and durations

Phase	Altitude	Mach #	Duration
1	0	0	NA
2	0	0	1
3	0	0.04	9
4	0	0.04	1
5	50	0.41	3
6	275	0.75	25
7	350	0.78	NA
8	150	0.71	25
9	100	0.54	25
10	100	0.45	3
11	0	0.3	1
12	100	0.54	50
	[FL]	[-]	[min]

5.1.1.3 Definition of criticality levels

Table 21: Criticality level definition

	Description of applicable failures	Degradation scheme	Example
Normal	No failure	None	-
Level I	Failure configuration which should not stop the aircraft from getting destination (dispatchable failure)	Slight degradation of non-vital functionalities dedicated passenger comfort is acceptable. Degradation should be transparent on aircraft performance	- Single generator failure - Failure of an ECS pac
Level II	Failure of (a) critical element(s). This failure leads to aborting the mission	Elimination of most non-vital functionalities dedicated to passengers is acceptable. Slight degradation of aircraft is acceptable.	- Failure affecting less than 50% of propulsive capability
Level III	Failure threatening the survival of the aircraft. This failure leads to aborting the mission.	Elimination of all non-vital functionalities. Significant passenger discomfort is acceptable. Temporary but significant degradation of aircraft performance is temporarily acceptable.	- Loss of power plant - Total loss of propulsive power

The criticality scale is introduced to classify failures. This scale is described by Table 21. Each failure configuration considered in this project will be classified under one of the three criticality level. Depending on the classification of these failures, the mission sequence may be modified. If any failure events classified as level I (dispatch-able failure) occurs, the mission sequence presented earlier is maintained. On the other hand if a failure event classified in Level II or III is realized, the original mission is susceptible to be aborted. The sequence of events in the aborted mission will vary depending on when the failure event occurred. The following paragraphs will define the alternate mission sequences which may be encountered. These sequences are defined and represented based on methods introduced in the methodology section of the thesis. It will consider each flight phase/failure status as states and the failure events as transitions between these states.

5.1.1.3.1 Failure on Ground (phases 1, 2 and 3)

If any failure event occurs before the take-off phase (phase 4), the mission is aborted and terminated.

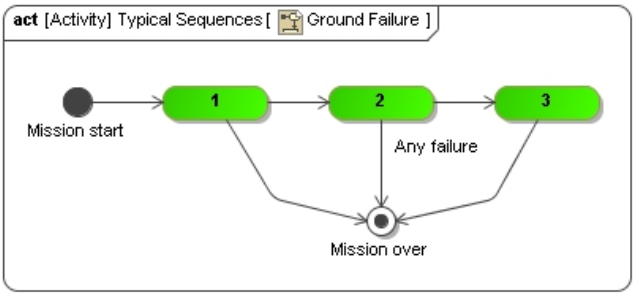


Figure 138: Phase 1, 2 and 3 failures mission sequences

5.1.1.3.2 Failure at Take-off (phase 4)

If a level I failure occurs during take-off, the mission will proceed as indicated in Figure 124. If failure events of criticality levels II or III occur, the aircraft will go around and perform an emergency landing. This alternate mission is represented by the sequence of an emergency take-off (phase 4) followed by an emergency landing (phase 11). These alternate missions are represented in the following figure:

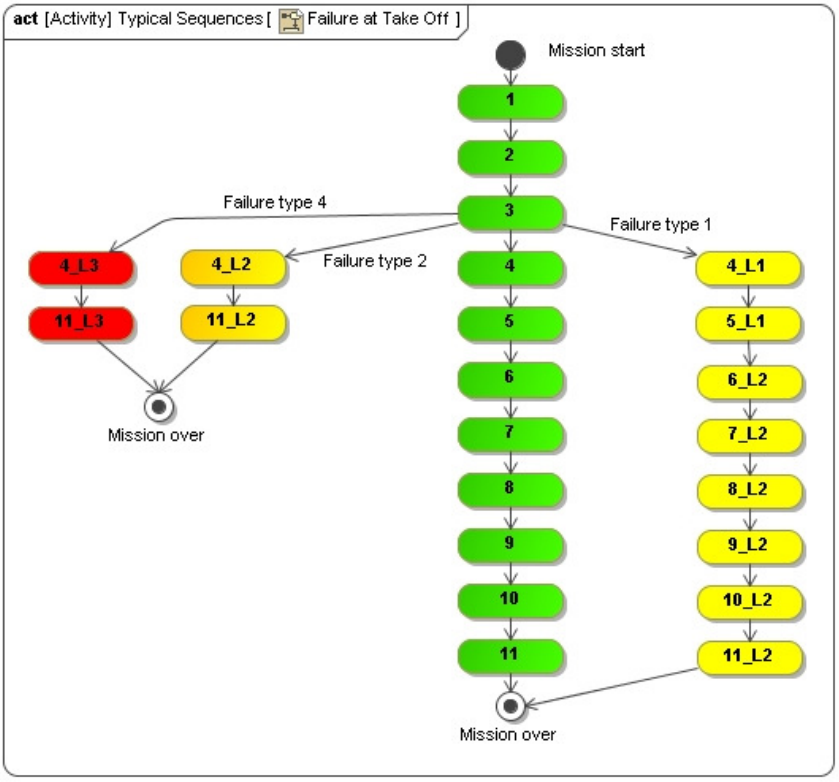


Figure 139: Phase 4 failures mission sequences

5.1.1.3.3 Failure in Initial Climb (phase 5)

If a level I failure occurs during initial climb, the mission will proceed as indicated in Figure 124. If failure events of criticality levels II or III occur, the aircraft will go around and perform an emergency landing. This alternate mission is represented by the sequence of an initial climb (phase 5) which will represent the go around downwind segment and an emergency landing (phase 11). These alternate missions are represented in the following diagram:

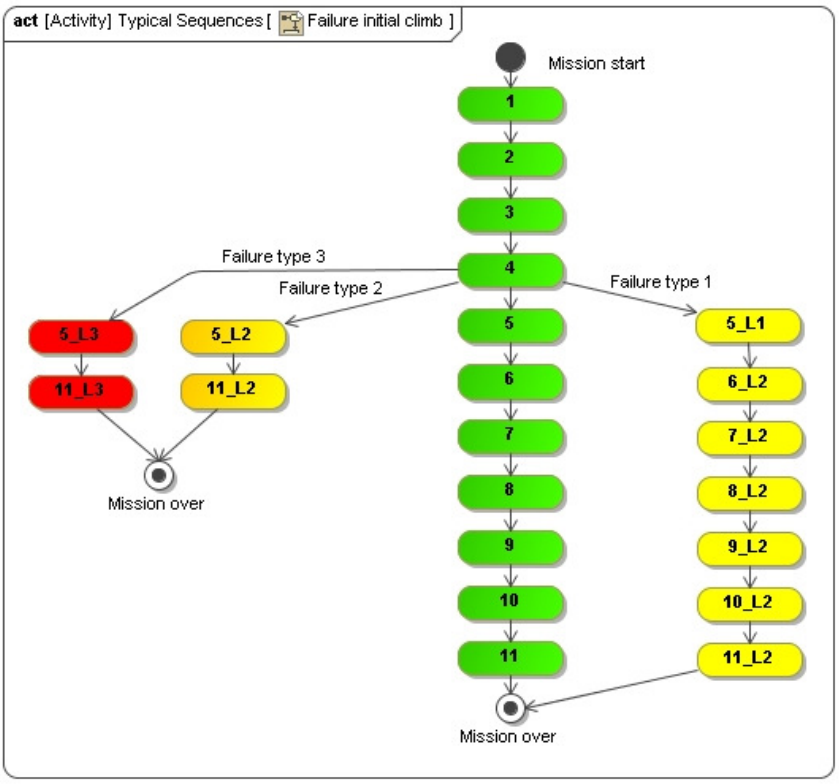


Figure 140: Phase 5 failures mission sequences

5.1.1.3.4 Failure between Final Climb and Initial Descent (phase 6, 7 and 8)

If a level I failure occurs during any of these phases, the mission will proceed as indicated in Figure 124. If failure events of criticality levels II or III occur, the aircraft will proceed to the nearest airport to perform an emergency landing. In phase 4 and 5, it was certain that the aircraft is still within the proximity of the take-off runway. In phases 6, 7, or 8 there are no certainty with regard to the immediate proximity of a runway. Therefore if a critical failure occurs in these phases, a travel phase to the emergency runway must be accounted for. Therefore this alternate mission is represented by the sequence of an initial descent (phase 8) which will represent the descent from the altitude at which the failure occurred down to 10,000 ft where the emergency ferry will occur (phase 12). Once in proximity of the alternate airport, the emergency final approach and landing will be represented by phase 10 and 11. These alternate missions are represented in the following diagram:

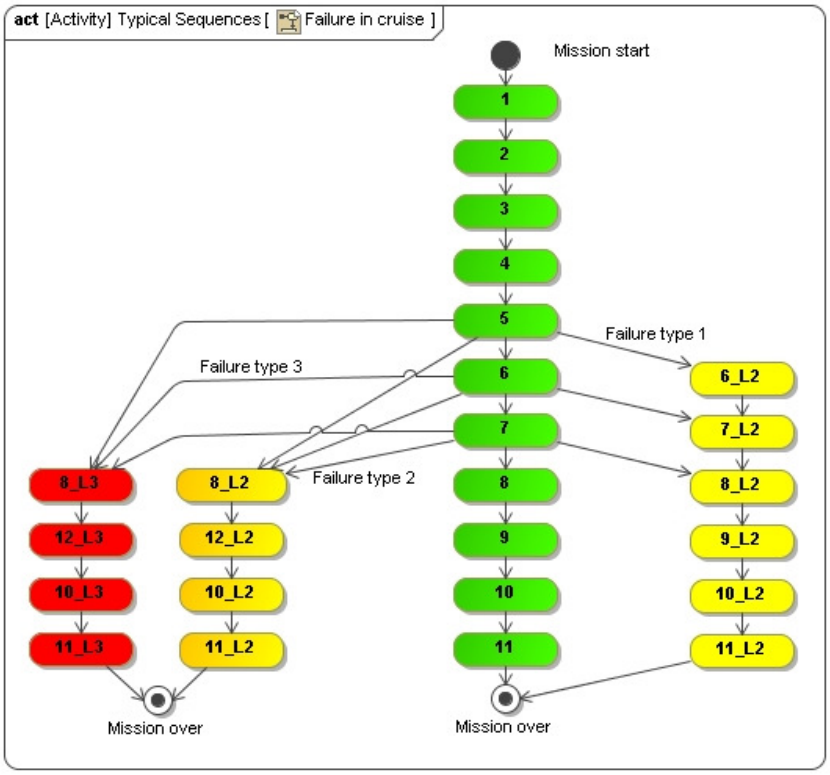


Figure 141: Phase 6, 7 and 8 failures mission sequences

5.1.1.3.5 Failure in Holding and Final Approach (phase 9 and 10)

If a level I failure occurs during any of these phases, the mission will proceed as indicated in Figure 124. In phase 9 and 10 it is assumed that the aircraft is within the vicinity of destination airport. For that reason, no ferry phase is required. This alternate mission is represented by a direct transition to the approach phase and landing (phase 10 and 11 respectively).

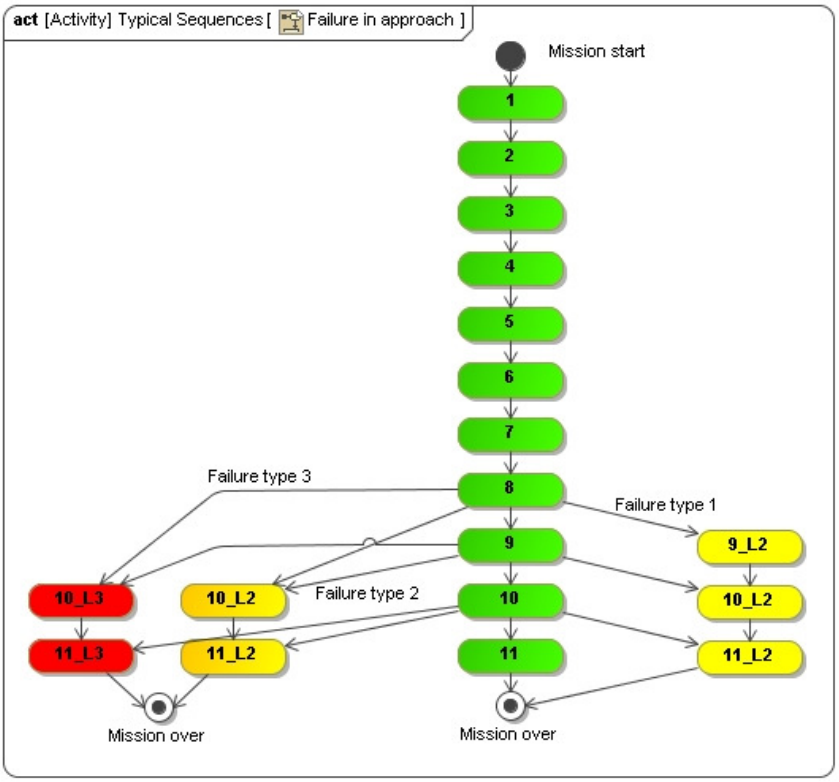


Figure 142: Phase 9 and 10 failures mission sequences

5.1.1.4 Elicitation of Requirements

Previously twelve flight phases were defined. In addition to these flight phases, four levels of criticality were formulated. Crossing these two mission parameters yielded 48 types of flight conditions. Based on the possible sequences formulated above 15 of these 48 scenarios do not need to be characterized. For instance, it is not necessary to

consider failures at flight phases before take-off (see Figure 138). Based on this logic 33 possible scenarios must be considered. These scenarios are shown in Table 22.

Table 22: Overview of valid combinations

Failure Level	Prk 1	St 2	Tx 3	TO 4	CL1 5	CL2 6	CZ 7	HL 9	Dsc 8	App 10	LD 11	Fer 12
0												
1												
2												
3												

	Valid combination
	Invalid combination

Based on information provided by EADS and Airbus S.A.S., each of the sub-functionalities was elicited for their requirements in each of these combinations. More details about the quantification of the boundary requirements (requirements associated with boundary functions) is presented in Appendix A.

5.1.2 **Discussion of Results**

The functional breakdown's ability to decompose and organize mission needs was demonstrated along with the ability to quantify requirements provided by mission parameterization. The use of criticality levels allows for the definition of requirements under failure scenario. This feature enables an architecture independent description of the requirements. A collateral effect of this method is to help the architect understand and isolate implicit assumptions about functional degradations under failure scenarios. If we are used to very specific degradation profiles entrenched to the limitations of a classical architecture, having to define generically what the fundamental need is (which functions one can really afford to degrade under critical failure scenario) provides an important insight into the fundamental needs of the mission.

Several minor reservations should be observed. These reservations do not compromise the usefulness of this method in dealing with conceptual investigations of aircraft power system architectures. They are described in the following paragraphs.

5.1.2.1 Temporal Decomposition

While the mission can be formulated in the form of mission sequences which define the order of realization of the scenarios, it will not qualify the transient behavior of the requirements and it will quantify the mission requirements associated with each state independently. The lack of transient behavior could possibly be addressed within the frame provided by the methodology proposed in *Hypothesis 1*. The first approach would be to use transient boundaries (as presented in the interface standard by the United States Department of Defense [74] and implemented by Khozikov et al. in reference [75] and Phan in his PhD dissertation [76]). These boundaries provide an envelope within which power requirements (transient peak load) and characteristics (voltage or frequency) are allowed to fluctuate. This approach provides information about the dynamic behavior of the requirements within a quasi-static framework. Indeed instead of using a deterministic approach to the characterization of a functional requirement as a function of time, this approach provides boundaries which qualify extreme transient scenarios.

Also this scenario by scenario approach does not consider the fact that a function can be performed at any point over a sequence of flight phases. For lack of a better illustration, one can consider functions like “cooking the meals”. If this function were performed on the ground during taxi, it would no longer be necessary during climb. In the present formulation of the mission, the power implied by the “preparation of the meals” is imposed on multiple scenarios (despite the fact that it is necessary to perform this function only once).

5.1.2.2 Functional Decomposition

As mentioned earlier, the functional breakdown provides a convenient decomposition and classification of the needs. This decomposition does, however, imply certain specific assumptions. In this context, the decomposition isolates the performance of each function with respect to one another. For instance, although the requirements on

propulsive function address this function directly, they could also be used for control purposes. In the context of a distributed propulsion concept, asymmetric thrust distribution could be used instead of control surfaces in order to provide maneuverability. Using a functional breakdown makes it difficult to capture, in a generic fashion, trans-functional requirements. It is able to capture these trades only by updating the formulation of the breakdown and/or the quantification of the functional requirements.

5.1.2.3 Criticality Level Classification

The criticality level approach implies general degradation profiles. The degradation profiles are defined in a generic manner. Any failure configurations classified under the same criticality level would be required to perform the same requirements. In cases where the failures are localized on a specific capability this formulation will impose functional requirements which may seem inadequate. In order to illustrate this situation let us consider the following example. The architecture considered is composed of two power plants and six electric fans. Let us consider a failure configuration where four of the six fans are inoperative (both power plants still operate normally). In this situation over 50% of the propulsive capability is lost, therefore this failure configuration is classified as a level III failure state. Based on the degradation scheme indicated by this criticality level, all functionalities non essential to the survival of the passenger are shed. In criticality level III the air conditioning functionality (provision of energy to the ECS) is degraded to a point where the cabin pressurization requirement is no longer enforced but oxygen masks are required instead. In this case, the pressurization is not considered as a function "essential to the survival of passenger" as long as the oxygen masks are provided. Therefore, in the failure configuration where propulsive capability is lost, the requirements associated with the air conditioning are degraded. This degradation in requirement occurs despite the fact that no loss affecting the power production capability (supporting the ECS) was observed. It is important to

note that the degradation profile defines constraints the sizing of subsystems. Therefore, in the situation described above, even if the power plants are not “forced” to provide ECS power in case of electric fan failure, it does not mean that they are not capable to provide this functionality in that situation. The purpose of the functional requirements is only to guaranty the minimally acceptable performance. But it does not mean that this minimal performance always correspond to the actual performance.

5.1.2.4 Conclusions of Experiment 1

The observations made in this example support the validity of *Hypothesis 1*.

Hypothesis 1: The functional requirements can be characterized by defining scenarios based on flight phases, flight conditions and criticality depths.

The method based on this hypothesis can successfully capture a commercial transport aircraft mission. Several limitations were indicated but none of these limitations compromise the fundamental ability to quantify the minimum functional requirements in an architecture-independent fashion.

5.2 Experiment 2: Sizing of subsystems

In this thesis, the Coordinated Optimization method was proposed to implement the sizing and optimization of architecture systems. This method was formulated based on a set of four hypotheses. The validation of the Coordinated Optimization method is structure around two experiments. Experiment 2 is the first of these two experiments. It will focus on the testing of the proposed subsystem sizing hypothesis while experiment 3 will investigate hypotheses dedicated to the architecture optimization.

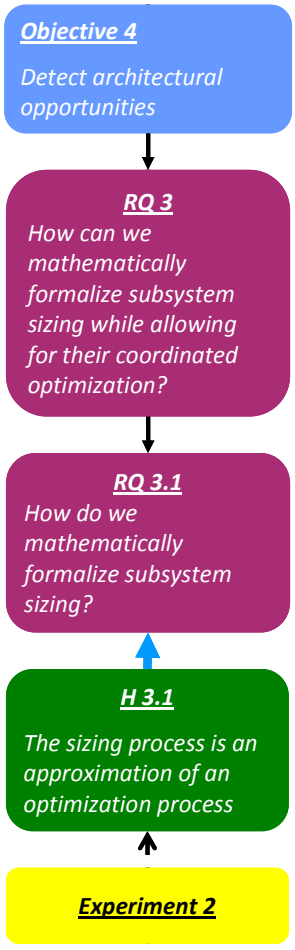


Figure 143: Experiment 2 overview

In this thesis a subsystem centric approach to architecture analysis is proposed. This approach facilitates both the integration of technological knowledge while allowing for the composition of architecture analysis. This approach is based on the sizing process

of subsystems. In order to improve the quality of subsystem sizing processes *Hypothesis 3.1* was proposed. It introduces the idea that the sizing is best represented by an optimization process. This experiment will verify and demonstrate the validity of the hypothesis formulated previously. This experiment will be based on the comparison of two approaches to sizing: The first approach is the functional regression approach. This approach assumes that based on the functional requirements only, the subsystem attributes can be derived directly. The second approach is the optimizer-based sizing model. In this context, the attributes of the subsystem are defined via an optimization process based on an objective function (relating the system-level objective to subsystem attributes) and functional requirements.

In order to compare the effectiveness of the two approaches, this experiment will observe the following two aspects of the sizing process. First it will consider the exploration of the trade-offs internal to the subsystem. The ability perform them is essential to *Objective 4* (*detecting architectural opportunities*). The second aspect concerns the practicality of the model. In order to be practical, the resulting model must be both reasonably fast and reliable in finding a sized solution. This aspect is necessary to the fulfillment of *Objective 3* (*accelerate the turn around in architecture concept analysis*)

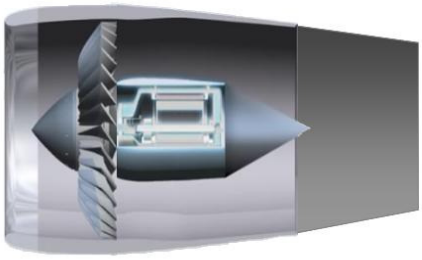


Figure 144: Graphical representation of the electric ducted fan

The test case used in this experiment will be an electric ducted fan. This subsystem is represented in Figure 144. It provides thrust in exchange for electric energy. This type of subsystem provides an interesting test-case for the experiment for several

reasons. First of all, this subsystem provides clear trade-off opportunities between weight and efficiency. Larger fan are heavier but more efficient (increased propulsive efficiency). Secondly, it was possible to constitute a physics based model with the assistance of Philippe Masson from the Advanced Magnet Lab (Palm Bay, FL). With this physical analysis base, it is possible to perform realistic analysis similar to what would be necessary for an industrial development. Also, this enables physical insight into the sizing conclusions provided by the alternative approaches.

5.2.1 **Deployment of the Optimizer-Based Sizing Methodology**

5.2.1.1 Creation of the Optimizer-Based Sizing Model

The model was developed in summer 2009 with the assistance of Taewoo Nam (Aerospace System Design Laboratory) and Philippe Masson (Advanced Magnet Laboratory). A description of the model is provided in Appendix B. It is based on an aerothermodynamic analysis estimating the thrust performance of a fan/duct assembly subjected to power, torque and speed constraints from the motor. The motor performance was estimated using a model developed by Philippe Masson [77-78]. These models were used to perform the following optimization problem:

$$\underset{\overline{X}}{Min} \quad \gamma_1 \frac{m_{engine}\left(\overline{X}\right)}{m_{engine-baseline}} + \gamma_2 \frac{diameter\left(\overline{X}\right)}{diameter_{baseline}} + \gamma_3 \frac{1-\eta_{CZ}\left(\overline{X}\right)}{1-\eta_{CZ-baseline}} + \gamma_4 \frac{PowEreq\left(\overline{X}\right)\Big|_{\max}}{PowEreq\Big|_{\max-baseline}} \qquad \textbf{(21)}$$

Subjected to:

- $ThrustReq_k - ThrustCap\left(\overline{X}\right)_k < 0$ for all $k \in [1, K]$

Where:

- $X(1)$: $Afan$ – Fan face area [m2]
- $X(2)$: ro – Mean armature radius [mm]
- $X(3)$: Arm – Machine shape factor [La/ro]

- $PowEreq\left(\overline{X}\right)\Big|_{\max}$ – Max electric power required in mission
- $\eta_{CZ}\left(\overline{X}\right)$ – Efficiency in cruise
- $ThrustReqk$ – Thrust required in scenario k
- $ThrustCap\left(\overline{X}\right)_{\mathrm{k}}$. – Max thrust available in scenario k

The sized solution for the electric fan corresponds to the design which optimizes the problem formulated above. The aerothermodynamic and motor limit model allow for the evaluation of the ThrustCap variable along with the evaluation of the electric fan attributes (mass –m-, diameter, efficiency at cruise and power requirements to operate).

5.2.1.2 Addressing Challenges

In order to have a practical sizing process it is necessary to have a model which is both reasonably fast and reliable. When executing the model above directly, several challenges were observed.

5.2.1.2.1 Difficulties Finding Feasible Space

The first challenge pertained to the reliability of the sizing process. This lack of reliability was observable via two specific symptoms. The first symptom was the difficulty finding feasible space in the beginning of the optimization process. This symptom was primarily due to the fact that it is not possible to evaluate the objective function at infeasible solution. When using physics based modeling, it is not always possible to evaluate the attributes of infeasible designs. For instance, if a fan is expected to provide 10kN of thrust but has a capability limited to 5kN, it is not possible to evaluate the efficiency of such a machine. Because of this limitation, the use of exterior penalty functions guiding the optimizer toward feasible space can not be implemented. In order to remedy this problem an ad-hoc scaling approach of the design parameters was used. This scaling approach would pre-size the fan and motor in a serial fashion and make sure that together they constitute a propulsor capable of performing the requirements. Using this

approach allowed the deployment of an interior penalty function described in equation (22).

$$F\left(\overline{X}\right)=\begin{cases} OEC\left(\overline{X}\right)+P\left(\overline{X}\right) & \text{for } \overline{X} \text{ feasible} \\ OEC_{\inf} & \text{for } \overline{X} \text{ infeasible} \end{cases} \quad \textbf{(22)}$$

Where:

- $P\left(\overline{X}\right)=\dfrac{rp}{\max\left(ThReq\text{-}ThCap\left(X\right)\right)}$
- $\mathrm{OEC}\left(\overline{X}\right)=\gamma_1\dfrac{m_{engine}\left(\overline{X}\right)}{m_{engine-baseline}}+\gamma_2\dfrac{diameter\left(\overline{X}\right)}{diameter_{baseline}}+\gamma_3\dfrac{1-\eta_{CZ}\left(\overline{X}\right)}{1-\eta_{CZ-baseline}}+\gamma_4\dfrac{PowEreq\left(\overline{X}\right)\Big|_{\max}}{PowEreq\Big|_{\max-baseline}}$
- $O(\mathrm{OEC}_{\inf}) >> O(\mathrm{OEC})$

5.2.1.2.2 Convergence to Optimum Solutions and Acceleration of Resolution

The physics based model used in modeling the performance relies on numerical iterative processes. These processes attempt to solve for equations which can not be solved formally. The convergence of these processes is defined based on convergence criterions which limit the computational expense associated with the resolution of equations. The convergence criterion will tune both the time necessary to execute the model and level of resolution of the model.

Search-direction-based optimization algorithms search the local topology of the objective function around the point currently considered. If in doing so, the optimizer is exploring within the level of resolution of the model, it will be observing noise rather than the general topology (which would enable the identification of the optimal search direction). Although the numerical resolution of the physics-based model is able to capture the large scale responses, the local behavior of these responses can be noisy and

may prevent the optimizer from selecting the correct search direction. Because of this problem, the ability of the optimizer to find the global optimum is strongly limited[2]. In order to address this issue, three solutions could have been considered:

- Decreasing the convergence criterions of the physics-based models therefore increasing the resolution of the model
- Using stochastic optimization which does not rely on search direction
- Using regressions capturing the large-scale behavior of the physics-based model without local-scale interferences.

The first two options were inappropriate because they would have greatly increased the time necessary to find an optimum solution. As a result, the only option viable was the use of regression to "smooth-out" the behavior of the model while accelerating its resolution

5.2.1.2.3 Preservation of physical constraints in a regressed model

The advantage of using a physics based model is that it is capable of capturing the discontinuities in the design space. The situation shown in Figure 145 provides an example for this capability.

[2] I would like to thank Chung Hyun Lee for his help in understanding this optimization problem

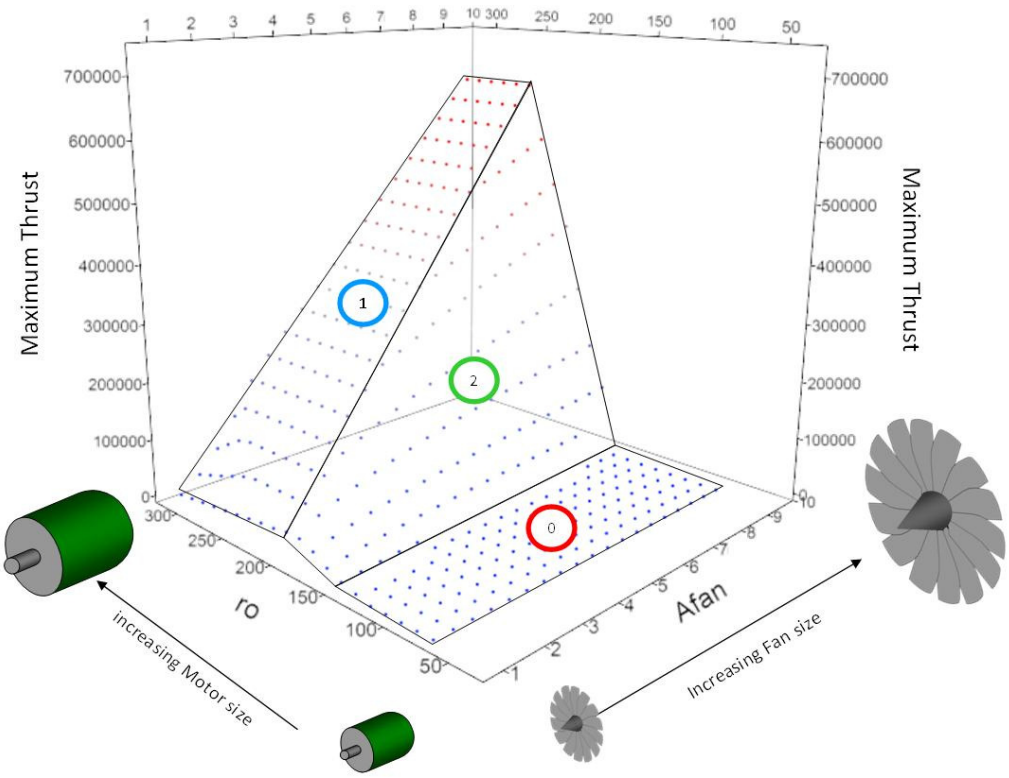


Figure 145: Thrust capability as a function of fan and motor sizes

This figure shows the ability of an electric fan to produce thrust as a function of the "sizes" of its components (fan and motor). In this example we can see that the capability will behave very differently in different regions. In some regions, it will not be able to produce thrust (region 0) but in others, its capability will be limited in different ways. This difference results from the fact that in region 1 the fan will be saturated (the motor is "over-sized" compared to the fan) while in region 2 the motor will limit the overall capability. Naturally, the physics-based model will capture this discontinuity.

A regression model which attempt to capture the general trend of the response does not behave well with this form of discontinuities. When attempting to fit this response, fitting the entire space was not possible. A low reaction regression model (i.e neural nets with few nodes) will fail to capture the discontinuities while high reaction regression models will tend to over fit the response hence artificially creating singularities. In a context where the regression model is used for optimization these

singularities are a definite showstopper (For more information, please refer to the electric fan Appendix B).

In order to create a surrogate which respects the discontinuities of the original responses while avoiding creating singularities, a method for partitioning the design space based was defined and implemented. This method consisted in detecting the boundaries of regions with a specific dominating phenomenon (saturated fan versus saturated motor). A first set of regressions was used to fit the coordinates of the boundary between these regions; a second set was used to fit the responses in each region (one regression for each region). Finally, these regressions were associated, with "if" statements integrating these local regressions into a single regression characterizing the entire design space.

Using this approach allowed formulating a model which was fast, smooth and physically accurate in its discontinuities. This regression enabled a robust and rapid optimization process, which rendered the optimizer-based sizing model both fast and reliable.

5.2.2 **Presentation of a Functional Regression Sizing Model**

As described in chapter 4, the functional regression sizing model will formulate a sizing solution which is based only on the functional requirements. As a result, the functional regression corresponds to an optimizer-based sizing model where the objective function is fixed. Therefore, in order to represent the behavior of a functional regression model, I will consider the response of the model described above for a fixed objective function.

5.2.3 **Comparison of Results and Conclusions about Sizing Methods**

5.2.3.1 Ability to Capture Subsystem-Level Trade-offs

5.2.3.1.1 Observation of Functional Requirement effects

In order to observe the effect of the amplitude of the thrust constraints, the requirements associated with a basic mission considered in the thesis were scaled and submitted iteratively to the model subjected to a fixed set of priority factors. Figure 146 presents this evolution for five different optimization priorities (each color represents points sized with a different priority). The five strategies and their respective color used to produce these lines are defined in Table 23.

Table 23: Notional optimization priorities

$$OEC(\overline{X}) = \gamma_1 \frac{m_{engine}(\overline{X})}{m_{engine-baseline}} + \gamma_2 \frac{diameter(\overline{X})}{diameter_{baseline}} + \gamma_3 \frac{1-\eta_{CZ}(\overline{X})}{1-\eta_{CZ-baseline}} + \gamma_4 \frac{PowEreq(\overline{X})|_{\max}}{PowEreq|_{\max-baseline}}$$

Optimization priority	γ_1	γ_2	γ_3	γ_4
Balanced	0.25	0.25	0.25	0.25
Power Optimized	0.25	0	0.25	0.5
Energy optimized	0.25	0	0. 5	0.25
Aggressive power optimization	0	0	0.5	0.5
Engine weight optimization	1	0	0	0

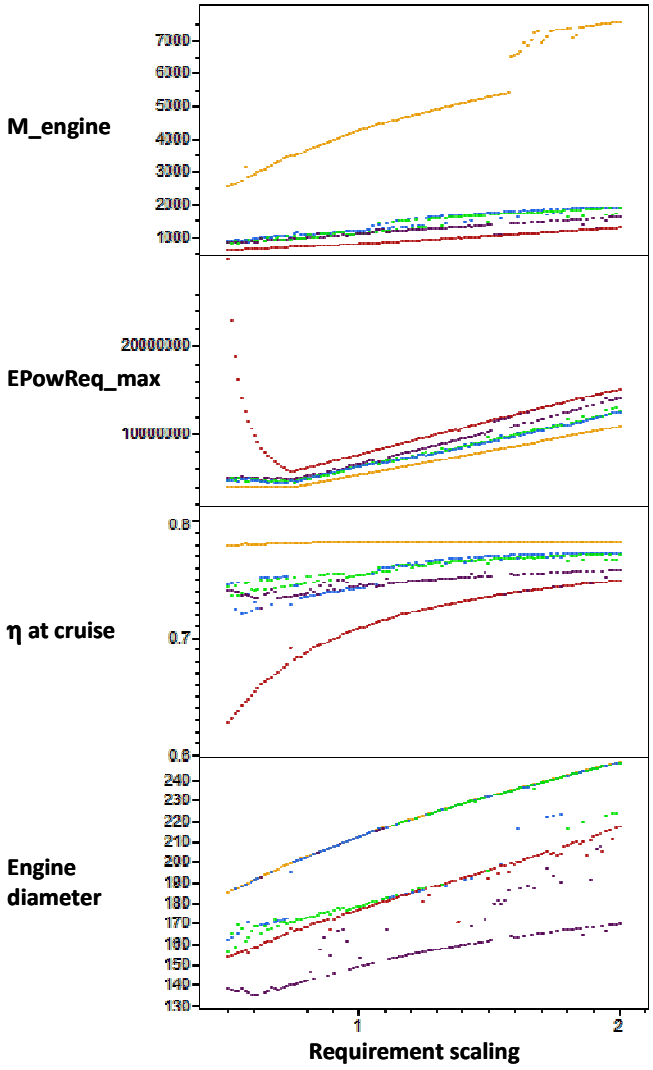


Figure 146: Attributes scaled by requirements

From these plots, we can observe the typical evolution of the engine attributes as a function of the thrust requirements. These plots highlight the fact that the growth of the engine is strongly conditioned by the prioritization strategy. If a functional regression was used, the sizing model would only capture one of the colored lines. Therefore, the diversity in solutions would not be captured by the model.

If we observe the spread between the lines in Figure 146, we can see that depending on the optimization strategy used in the sizing of the subsystem, variation accounting in excess of 100% could be observed. This large variation results from the sizing trade-off of the fan. A larger fan will operate on a larger mass-flow which, for a given thrust level, will allow for higher efficiency. But this efficiency has a cost which is the additional weight induced by the over-sizing of the fan.

5.2.3.1.2 Observation of Optimization Strategy Effects

In order to further observe the effect of prioritization strategy, an exploration of the priority effects was performed. The objective function considered by the optimizer (presented in Table 23) is composed of 4 parameters. The priority factors associated with each of these four parameters were explored by performing a full factorial DoE. Each experiment in the DoE corresponds to a specific optimization problem with identical constraints but with different objective functions. The following scatter plot represents the ensemble of sized design solutions which were returned by the model. Each point in this scatter plot represents a feasible solution (i.e. a design which meets the thrust required by the mission). The line in green represents the Pareto front.

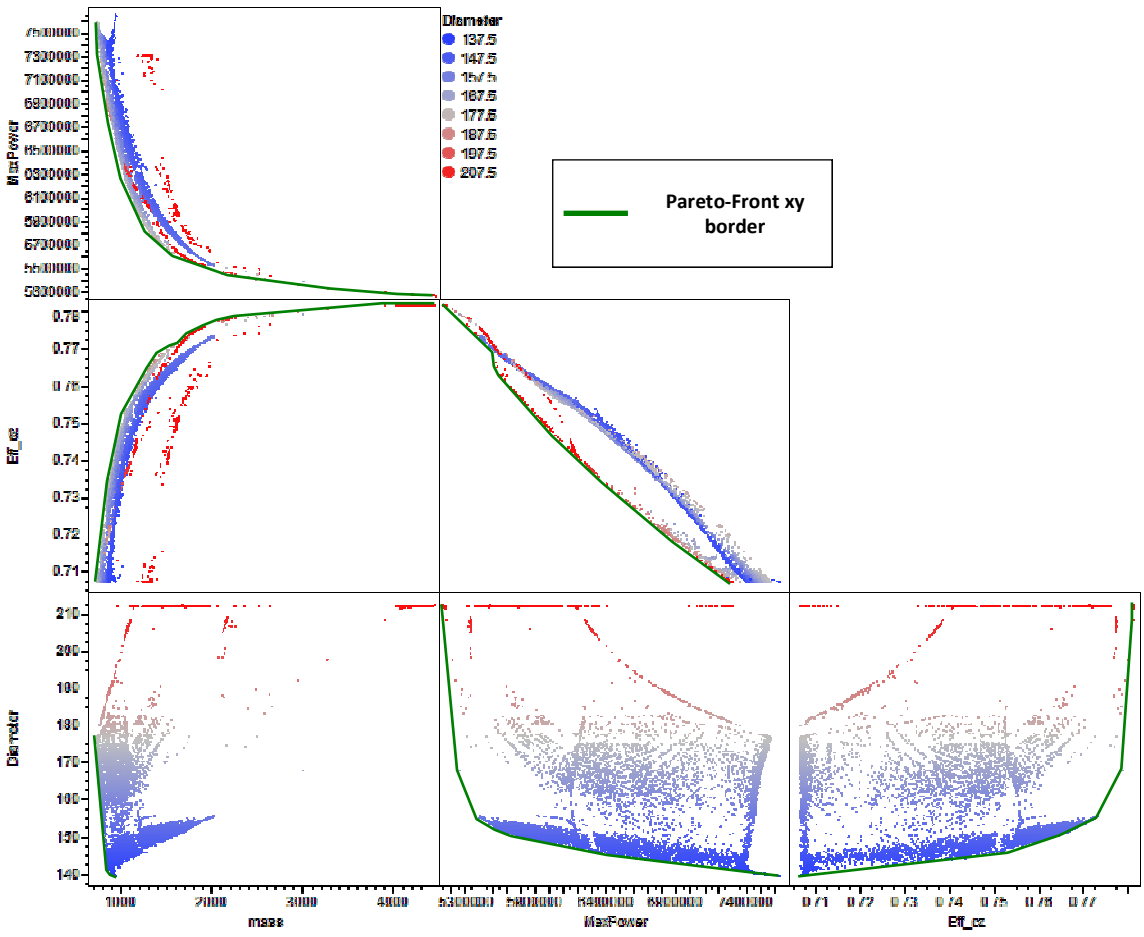


Figure 147: Overview of internal trade-offs

Figure 147 presents a number of design solutions to the same thrust requirements (i.e. aircraft mission). The great diversity in the attributes of the solutions presented in this figure highlights the need to perform an optimizer-based sizing process. The model proposed herein therefore does not only identify an engine design appropriate for the requirements; granted that the appropriate objective function is provided, it can identify the "best" electric ducted fan design for the architecture.

5.2.3.1.3 Experiment Conclusion about Quality of Sizing

From an architectural point of view, subsystem-level trade-offs can be critical to overall performance. In the case of the electric fan, the electric power received is the sizing requirement to the chain of power systems providing its electricity. Therefore, every additional watt that must be delivered to the electric fan will imply a weight

penalty to the architecture (larger generator, larger power plant, higher fuel burn). Thus, the trade-off between weight and efficiency is capital not only to the sizing of the subsystem (i.e. the electric fan) but also to the performance of the system (i.e. the aircraft).

In situations where a functional regression sizing model is used, the optimization strategy will be implicitly fixed (i.e. the estimations will follow one of the lines in Figure 146). If we are dealing with a well-known architecture, where the ideal subsystem optimization strategy is known, this approach may still lead to an optimal conclusion. However, functional regression should not be used to size subsystems offering very diverse forms of solutions depending on their optimization strategy. This conclusion is especially true for situations where such subsystems are integrated in new architectures.

The absence of consideration of subsystem-level trade-offs by the functional regression model highlights its profound inability to fully realize the optimization potential of the subsystem. Hence, it can be concluded that not taking into account the optimization strategy will yield sizing conclusions which are sub-optimal and which will lead to conservative assessments of new architecture.

5.2.3.2 Practicality for the Development of Sizing Models

As presented previously, implementing an optimizer-based sizing approach is a complex process due to both the use of physics based models and the setup of the optimization. The first challenge is related to the impossibility in evaluating the objective function outside of feasible solutions (due to the lack of physical meaning of infeasible physical attributes). This problem can be addressed by an ad-hoc approach to pre-sizing of the subsystem, and the use of an interior penalty function. The second problem is related to the lack of reliability of the optimization problem based on the physics-based model. This problem was addressed by substituting the physics-based model by a regression which enabled a stabilization of the optimization process.

In order to compare the implementation process of the optimizer-based sizing model to the process of implementing a functional regression model, I will assume that creating the later would only correspond to performing a root mean square regression over a set of pre-defined electric fan designs (i.e. regressing on one of the line presented in Figure 146). Therefore, the complexity associated with the creation of the functional regression is considerably simpler than the optimizer-based approach.

The difficulties observed during the elaboration of the sizing model of the electric fan were a fairly typical demonstration of the challenges associated with the elaboration of optimizer-based sizing models. Based on this , we can draw the following conclusion on the practicality of the definition of an optimizer-based sizing model. The optimizer-based sizing approach can only be formulated by an expert designer. In order to produce both a practical and effective model, it is necessary to use some heuristics (pre-sizing, and tuning the regression). On the other hand, the elaboration of a functional regression model only requires regression over a set of data and does not require technical expertise. Therefore, creating an optimizer-based sizing model has a cost. Optimizer-based sizing provides the ability to explore subsystem-level trade-offs, but this benefit must be carefully balanced with the additional cost associated with its development.

It can therefore be concluded that the optimizer-based sizing approach (*Hypothesis 3.1*provides a solution to *Research Question 3.1* (*How can we mathematically formalize subsystem sizing while allowing for their coordinated optimization?*) and supports *Objective 4* (*Detect architectural opportunities*). But it is also important to observe that this approach has a development cost that must be carefully balanced with its advantages.

5.3 Experiment 3: Coordinated Optimization of Subsystems

5.3.1 **Overview of the experiments**

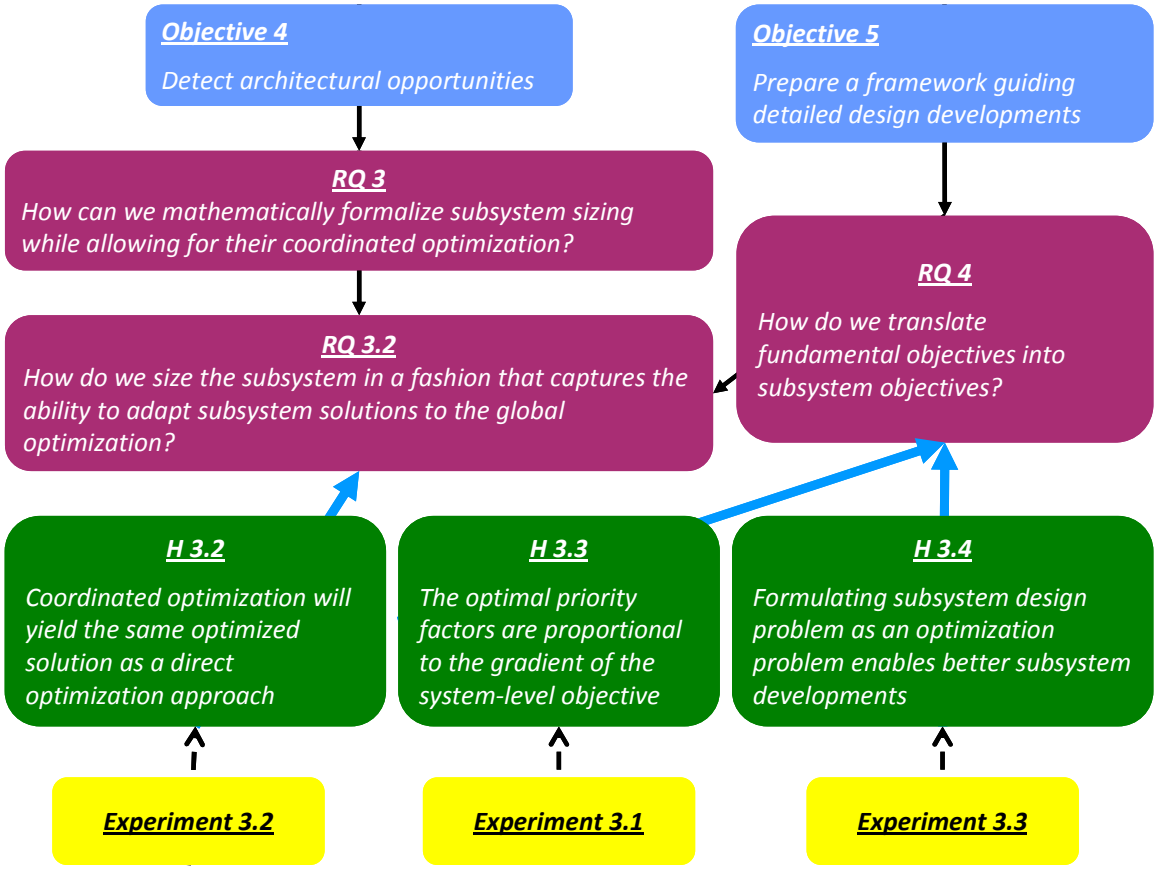


Figure 148: Experiment 3 overview

Experiment 3 attempts to validate several key points motivating or supporting the coordinated optimization scheme proposed in *Hypotheses 3.1, 3.2, 3.3* and *3.4*. Previously in Experiment 2, the agility of the optimizer-based sizing approach was demonstrated (*Hypothesis 3.1*). In experiment 3, we shall provide evidence of the architecture-level optimizer ability to both correctly steer subsystem sizing (*Research Question 3.2*) and to contribute to the formulation of the subsystem design problem in preparation for the preliminary and detailed design phases (*Research Question 4*). In order to produce this evidence, the experiment will demonstrate the validity of *Hypotheses 3.2*, *3.3* and *3.4* using three sub-experiments focused on specific aspects. The

first element of this experiment (3.1) will provide evidence that optimal priority factors are proportional to the gradient of the system-level objectives with respect to the subsystem-level attributes (*Hypothesis 3.3*). Experiment 3.2 should demonstrate the ability of the coordinated optimization to converge to an optimal architecture solution, hence demonstrating the validity of *Hypothesis 3.2*. The third aspect of this experiment will test *Hypothesis 3.4*. It will do so by comparing the performance of architectures on objective-based development to others via target-driven developments.

5.3.2 **Experiment 3.1: Formal Analysis of the Coordinated Optimization**

In order to understand the mathematical meaning of priority factors, a formal analysis is proposed. This analysis attempts to provide analytical evidence supporting *Hypothesis 3.3*.

5.3.2.1 Formal representation of the multi-level optimization problem

In order to support this analysis we shall consider an arbitrary architecture composed on N subsystems. This architecture is optimized based on the design parameters $\left(\overline{X_{\bullet}}\right)$ of its subsystems for the minimization of an overall Figure of Merit $\left(FoM\right)$. The fundamental problem which correspond a simplified version of equation (1) from Chapter 2 is expressed as:

$$\underset{\overline{X_{\bullet}}}{Min} \quad FoM \tag{23}$$

The subsystem design variables $\left(\overline{X_{\bullet}}\right)$ impact the system-level figure of merit $\left(FoM\right)$ via the subsystem-level attributes $\left(\overline{Att_{\bullet}}\right)$. The design variables and subsystem attributes associated with a given subsystem *n* are designated by the vectors $\overline{Att}_n$ and $\overline{X}_n$. The attribute vector is composed of M_n elements. Hence $\overline{Att}_n = \left[Att_{1,n},...,Att_{m,n},...,Att_{M_n,n}\right]$. The ensemble $\left(\overline{Att_{\bullet}}\right)$ and $\left(\overline{X_{\bullet}}\right)$ express the ensemble

of the $\overline{Att}_n$ and $\overline{X}_n$ vectors dedicated to the *N* subsystems composing the architecture. Therefore:

$\overline{Att}_\bullet = \left\{\overline{Att}_1,...,\overline{Att}_n,...,\overline{Att}_N\right\}$ and $\overline{X}_\bullet = \left\{\overline{X}_1,...,\overline{X}_n,...,\overline{X}_N\right\}$

FoM , $\overline{Att}_n$ and $\overline{X}_n$ are related as follow:

System synthesis relationship:

$$FoM = f\left(\overline{Att}_\bullet\right) \tag{24}$$

Subsystem description:

$$\overline{Att}_n = f\left(\overline{X}_\bullet, \overline{Att}_\bullet\right) \tag{25}$$

Hence we can see that the fundamental problem can be reformulated as:

$$\underset{\overline{X}_\bullet}{Min} \quad FoM\left(\overline{Att}_\bullet\left(\overline{X}_\bullet, \overline{Att}_\bullet\right)\right) \tag{26}$$

In chapter 2 we have recognized the fact that the architecture development is based on clustered concurrent engineering developments where subsystems are developed quasi independently from each other. This observation led us to *Objective 5* which requires a decomposition of the main system problem – as formulated in Equation (23) – into subsystem design problems which can be formulated theoretically as:

$$\underset{\overline{X}_n}{Min} \quad FoM\left(\overline{Att}_n\left(\overline{X}_n, \overline{Att}_\bullet\right)\right) \tag{27}$$

Subsystem-based developments are motivated by the need to simplify the analysis and the interactions between the subsystem and its architectural context. The designer of subsystem "*n*" wants to optimize FoM , based on the design variables pertaining to this subsystem $\overline{X}_n$. Within the context of his subsystem, the attributes of subsystems external to his own will constraint the performance of his own subsystems. Therefore, external attributes $\left(\overline{Att}_\bullet\right)$ will be considered as fixed and are noted as $\overline{Att'}_\bullet$. In order to represent this approximation, the objective function in Equation (26) must be reformulated as an approximation noted as $F\tilde{o}M$. In order for this approximation to be valid, the process of

minimizing $F\tilde{o}M$ and FoM must be mathematically equivalent. This is true if and only if $F\tilde{o}M$ and FoM are linearly proportional with respect to $\overline{X}_n$.

$$\underset{\overline{X}_n}{Min} \quad FoM\left(\overline{Att}_n\left(\overline{X}_n, \overline{Att}\bullet\right)\right) \Leftrightarrow \underset{\overline{X}_n}{Min} \quad F\tilde{o}M\left(\overline{Att}_n\left(\overline{X}_n, \overline{Att'}\bullet\right)\right) \tag{28}$$

$$\text{Iff } FoM\left(\overline{Att}_n\left(\overline{X}_n, \overline{Att}\bullet\right)\right) = C_0 + C_1 \times F\tilde{o}M\left(\overline{Att}_n\left(\overline{X}_n, \overline{Att'}\bullet\right)\right)$$

$$\text{With } \{C_0, C_1\} \in \Re^2 \text{ (i.e. } C_0, C_1 \text{ are independent from } \overline{X}_n$$

5.3.2.2 Taylor Series Expansion of the System-Level Objective

In order to compare $F\tilde{o}M$ and FoM, let us first itemize the relationship between FoM and the subsystem attributes. To do so we shall consider the Taylor series expansion of FoM with respect to subsystem attributes $\left(\overline{Att}\bullet\right)$. The series centered at $\overline{Att'}\bullet$ is represented in the following equation (where HOT refers to the Higher Order Terms):

$$FoM\left(\overline{Att}\bullet\right) = FoM\left(\overline{Att'}\bullet\right) + \sum_{i=1}^{N}\sum_{j=1}^{M_i} \left.\frac{\partial FoM}{\partial Att_{j,i}}\right|_{\overline{Att'}\bullet} \left(Att_{j,i} - Att'_{j,i}\right) + HOT \tag{29}$$

In the context of the development of subsystem *n*, subsystem attributes pertaining to other subsystems can be considered as fixed. Therefore:

$$\overline{Att}_i = \overline{Att'}_i \text{ for } \forall i \in \{1,...,N\} \setminus n \tag{30}$$

Hence, we can reduce equation (29) to:

$$FoM\left(\overline{Att}\bullet\right) = FoM\left(\overline{Att'}\bullet\right) + \sum_{j=1}^{M_n} \left.\frac{\partial FoM}{\partial Att_{j,n}}\right|_{\overline{Att'}\bullet} \left(Att_{j,n} - Att'_{j,n}\right) + HOT \tag{31}$$

In order to identify the elements necessary to the equivalence between $F\tilde{o}M$ and FoM, we need to identify the terms dependent on the value of $\overline{X}_n$.

$$FoM\left(\overline{Att}_n\left(\overline{X}_n, \overline{Att'}\bullet\right)\right) = \quad C_0 + C_1\left[\sum_{j=1}^{M_n} \left.\frac{\partial FoM}{\partial Att_{j,n}}\right|_{\overline{Att'}\bullet} Att_{j,n}\left(\overline{X}_n, \overline{Att'}\bullet\right) + HOT\right] \tag{32}$$

5.3.2.3 Applying the Condition to a Weighted Sum Subsystem-Level Objective

Now let us assume that the approximation of FoM in subsystem n was formulated as the weighted sum described in the equation (33). In this equation, the weighting factors are noted $\gamma_{j,n}$. They correspond to what was previously described as the "**priority factors**riority factor".

$$\tilde{FoM}\left(\overline{Att}_n\left(\overline{X}_n, \overline{Att'}_\bullet\right)\right) = C_1 \sum_{j=1}^{M_n} \gamma_{j,n} \times Att_{j,n}\left(\overline{X}_n, \overline{Att'}_\bullet\right) \quad \textbf{(33)}$$

Then in order to have equivalence between the minimization processes of $\tilde{FoM}$ and FoM :

$$\sum_{j=1}^{M_n} \gamma_{j,n} \times Att_{j,n}\left(\overline{X}_n, \overline{Att'}_\bullet\right) = C_1 \sum_{j=1}^{M_n} \left.\frac{\partial FoM}{\partial Att_{j,n}}\right|_{\overline{Att'}_\bullet} Att_{j,n}\left(\overline{X}_n, \overline{Att'}_\bullet\right) + HOT \quad \textbf{(34)}$$

For $\overline{X}_n \in \Re^D$ (where D is the number of design variables for subsystem n)

If we assume that the higher order terms are negligible compared to the first order, the we can observe that this condition above is true if and only if:

$$\gamma_{j,n} = C_1 \left.\frac{\partial FoM}{\partial Att_{j,n}}\right|_{\overline{Att'}_\bullet} \text{ for } \forall \begin{cases} n \in \{1,...,N\} \\ j \in \{1,...,M_n\} \end{cases} \quad \textbf{(35)}$$

5.3.2.4 Validation of Hypothesis and Discussion

Using the observation made in equation (33), at the optimal design the priority factors will be proportional to the local derivative of the system-level objective with respect to the attributes of the subsystems. Since the gradient of the overall objective is based on the first order derivative, we can conclude that *Hypothesis 3.3* is valid.

The priority factors formulate the direction toward the steepest improvement of the overall system based on subsystem attributes. This information provides a subsystem-level objective expression which represents the architecture-level objective function. Therefore *Hypothesis 3.3* directly addresses *Research Question 5*.

Although this formal analysis validates the nature of the information provided by the priority factors, it also highlights several important limitations that must be clearly understood by the reader.

The first important limitation concerns the completeness of the information provided by the priority factor. A subsystem has a lot of attributes, we can not assign a priority factor to all attributes or the architecture level optimization would become unmanageable. But not all subsystem attributes are actively constraining or influencing the performance at architecture-level either. Therefore, not all attributes are necessary and it is the responsibility of the system architect (or expert) to make sure that the objective function used for sizing includes all the necessary parameters.

The second limitation concerns the scope of application of the information provided by the priority factor. The optimization direction they formulate is based on a Taylor series expansion which by nature is limited to a point in close proximity to the reference point. It is also important to note that the local derivatives provided by the priority factors also depend on variables outside the perimeter of the subsystem. Therefore, although the formulation of the priority factors decouples the optimization of subsystems, this decoupling must be monitored with care, since a modification external to the subsystem may change the impact of its attributes on the overall objective. Hence, it is important to make regular updates as subsystems progress in their degree of definition.

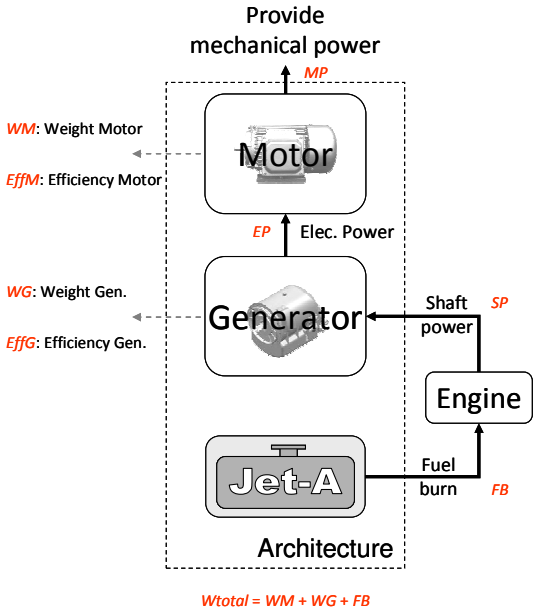
5.3.3 **Description of the Architecture Test-Case used in Experiments 3.2 and 3.3.**

The set of *Hypotheses 3.1, 3.2, 3.3* and *3.4* provide a formulation of the coordinated optimization approach. This approach has two main objectives: The first one is the identification of the optimal subsystems for the architecture (*Objective 4*) , the second is the formulation of a framework guiding detailed design developments (*Objective 5*) . Experiment 3.2 and 3.3 will investigate the performance with respect to

each of these objectives. They will be based on a notional study of a common simple architecture. This simple trade-off formulation and architecture concept were defined to display the same trade-offs that you would observe on a larger architecture. Using a simple architecture attempts to simplify the situation as much as possible without eliminating the phenomenon we intend to investigate. In this context, the simplicity of the architecture concept and its subsystem models allows us to have both a formal and an intuitive basis for the trade-off. This basis will provide a framework to validate *hypotheses 3.3* and *3.4*.

5.3.3.1 Description of the Simple Architecture

This simple architecture is composed of two elements, a motor and a generator. The generator is powered by an engine (fixed in size therefore not considered in the architecture sizing) which will consume fuel in order to support the provision of energy to the generator. The overall metric we wish to optimize is the total mass of subsystems (generator and motor) plus the fuel burn weight associated with their operation. Both the generator and motor can either be optimized on weight or efficiency. The sizing of this architecture implies an inter-subsystem trade-off between more efficient or lighter motors and generators. The architecture concept is described in Figure 149 and its problem definition is presented in the following equations.



Design problem:

$$\underset{X}{Min}\, Wtot \tag{36}$$

Subjected to:

$$Cap_m(X) \geq MP$$

$$Cap_g(X) \geq EP$$

$$EP = MP / EffM(X)$$

$$FB = Spe_{FB} \times SP$$

$$Wtot = WM + WG + FB$$

Figure 149: Simple architecture for example 3

The variable X is a vector composed of the motor and generator's design variables. The weight (W) and efficiency (Eff) are competing attributes (maximizing efficiency implies increasing the weight and vice versa). In order to represent this trade-off, equation (37) is used to represent the functional capacity Cap (MP in the case of the

motor and *EP* for the generator), as a function of the weight *W* and efficiency *Eff* (with *Eff* = 1 – *Loss*). The same equation format will be used for the motor and generator. The parameters a_1, b_1, p_1, a_2, b_2, and p_2 will be specific to each subsystem.

$$Cap = \left(a_1 \times Loss\big|_{\min} - b_1\right)^{p_1} \times \left(a_2 \times W\big|_{\min} - b_2 \times Cap\right)^{p_2} \quad \textbf{(37)}$$

$$Loss_{ideal} = b_1 \big/ a_1 \quad \textbf{(38)}$$

$$W_{ideal} = \frac{b_2 \times Cap}{a_2} \quad \textbf{(39)}$$

This form of equation provides a practical formula for the parameterized Pareto front. The asymptotes of the front defined in equations (38) and (39) show that the ideal efficiency is independent of the capacity while the ideal weight tends to increase with the power rating.

By transforming equation (37), we can reformulate the equation of the parameterized Pareto front, expressing subsystem weight as a function of its efficiency (or loss) and functional capacity. This transformation is provided in the following equation:

$$W\big|_{\min} = \frac{Cap^{1/p_2}}{a_2 \times \left(a_1 \times Loss\big|_{\min} - b_1\right)^{p_1/p_2}} + \frac{b_2 \times Cap}{a_2} \quad \textbf{(40)}$$

Using this form of equation will enable us to represent the front in a simple fashion that can be adapted easily by adjusting the parameters a_1, b_1, p_1, a_2, b_2, and p_2. The resulting from is represented in Figure 150.

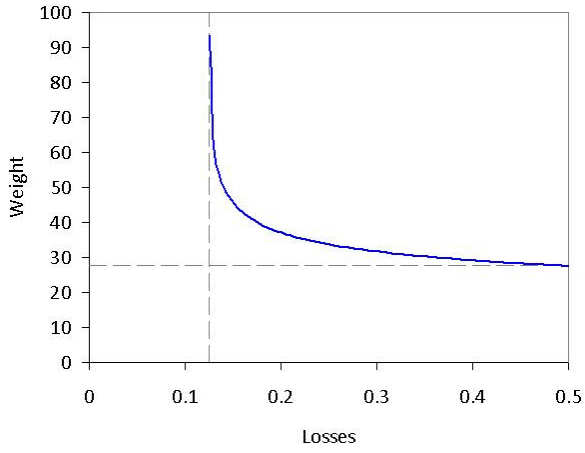


Figure 150: Notional Pareto front

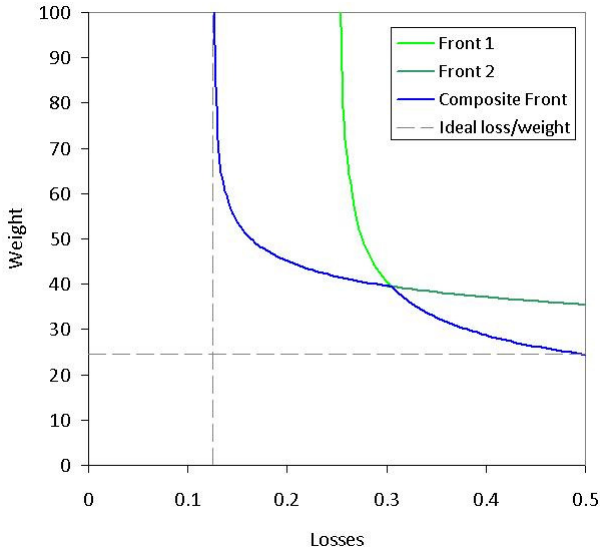


Figure 151: Notional non-convex Pareto front

The Pareto front represented in Figure 150 can be qualified as being convex. In order to better represent the nature of some subsystem trade-offs, the actual Pareto front was defined as being composed of two competing fronts. Each of these fronts is described by equation (37). The actual front is defined by selecting the minimum weight from the two fronts. An example of such a front is presented in Figure 151. This front is qualified as "non-convex". This shape of front is fairly typical in situations where a design variable is discrete. Often these trade-offs can be considered as internal architectural trade-offs to the subsystem. Since the methods proposed in this thesis are

expected to occur in the early design phases, multiple subsystem architectures may be considered for integration. Therefore, using a non-convex front could be considered a more realistic representation of the subsystem trade-offs, and shall also be considered in the following experiments

5.3.3.2 Introduction of experiment 3.2 and 3.3

In experiment 3.2 we will analyze the optimization process associated with the conceptual activities. This test consists in optimizing the architecture presented above with two different optimization strategies and compares the quality of the optimization process (which one provided the most optimal architecture, which one was the most practical and stable). In experiment 3.3, we shall observe the consequence of different subsystem design problem definitions. The effectiveness of the subsystem problem definition will be represented by making a simulation of subsystem developments. Hence, while experiment 3.2 and 3.3 both analyze the Coordinated Optimization (CoOp) method, experiment 3.2 will focus on the effects in the conceptual design phase, while experiment 3.3 will focus on the effects at the detailed design phases.

5.3.4 Experiment 3.2: Qualitative Analysis of Coordinated Optimization

5.3.4.1 Overview of the Test

Experiment 3.2 analyzes the optimization of the sample architecture proposed in this thesis by contrasting the behavior and results produced by the classical "All-in-One" optimization approach with those corresponding to a "coordinated" optimization approach.

5.3.4.1.1 All-in-One Optimization Representation

The "All-in-One" (AiO) approach is expected to represent the direct optimization of the system objectives based on subsystem design variables. In order to simulate this optimization approach we assume that any point on the Pareto front can be directly selected by the architecture level optimizer. In order to implement the selection, the optimizer provides two variables X_{motor} and X_{gen} between zero and one. These variables (which represent the motor and generator design variables) are related to the loss factor based on the following equation:

$$Loss(X) = Loss|_{ideal} + X \times \left(Loss|_{\max} - Loss|_{ideal}\right) \quad \textbf{(41)}$$

The value of the ideal loss is indicated by equation (38). The max loss value is chosen arbitrarily in order to put a bound on how inefficient the subsystem can be. By plugging the values for weight and loss of both the motor and generator into the constraints presented in equation (36) the overall weight can be determined. The corresponding weight value for the subsystem is determined using equation (40) and adapted in equation (42):

$$W(X) = \frac{Cap^{1/p_2}}{a_2 \times \left(a_1 \times Loss(X) - b_1\right)^{p_1/p_2}} + \frac{b_2 \times Cap}{a_2} \quad \textbf{(42)}$$

This optimization approach provides the architecture-level optimizer with the ability to choose directly the subsystem design. This approach represents the "traditional" approach to conceptual design were subsystem attributes or design variables are used to directly optimize the system and only requires an architecture-level optimizer. The optimization setup can be presented as:

$$\underset{x_1,x_2}{Min} \quad Wtot\begin{pmatrix} W_{motor}(X_1, Cap_{motor}), Loss_{motor}(X_1, Cap_{motor}),... \\ W_{gen}(X_2, Cap_{gen}), Loss_{gen}(X_2, Cap_{gen}) \end{pmatrix}$$

Subject to:

- $Cap_{motor} \geq MP$ **(43)**
- $Cap_{gen} \geq EP$
- $\{X_1, X_2\} \in [0,1]^2$

5.3.4.1.2 Coordinated Optimization

The coordinated optimization approach is based on two levels of optimization. The first, at the architecture-level, optimizes the priority variables. In this problem, there are two pairs of priority factors. Each pair defines the objective function of a specific subsystem. The first variable in the pair γ_1 quantifies the importance of attribute "*Loss*" to the optimization. γ_2 will do the same for W. These variables range from zero to one and the sum of each pair must be equal to one.

$$1 = \gamma_1 + \gamma_2 \text{ and } \gamma_i \in [0,1] \text{ for } i \in \{1,2\} \quad \textbf{(44)}$$

Note: Since γ_1 and γ_2 sum to one for each subsystem. We can say that for the problem considered in this experiment, the architecture-level optimizer controls two independent variables. The other two composing each pair can be deduced from the previous. In this experiment the two independent variables considered for coordinated optimization of the architecture will be $\gamma_{1\text{-motor}}$ and $\gamma_{1\text{-gen}}$.

At the subsystem-level, an optimizer will solve for the point on the front which optimizes the subsystem-level objective function. The subsystem-level objective function is defined in equation (45). This objective function used in this example is based on the overall evaluation criterion methodology:

$$Obj_{ss} = \gamma_1 \frac{Loss(X)}{Loss|_{ideal}} + \gamma_2 \frac{W(X)}{W|_{ideal}} \quad \textbf{(45)}$$

The values of loss and weight are related by equation (42). The minimization of the subsystem objective is performed by optimizing on variable X. In order to simplify this optimization process the solution to the minimization of equation (45) is defined formally. Based on equations (37) and (45), the loss value that will minimize the subsystem objective can be defined as:

$$Loss\big|_{best} = \left[\left(\frac{\gamma_1 \big/ Loss\big|_{ideal} \times a_2 \times p_2}{\gamma_2 \big/ W\big|_{ideal} \times a_1 \times p_1 \times Cap^{1/p_2}} \right)^{-\frac{p_2}{p_1+p_2}} + b_1 \right] \frac{1}{a_1} \quad \textbf{(46)}$$

It is important to note that in the coordinated approach, the architecture-level optimizer does not have direct control over the subsystem design variables. It steers the definition of the subsystem by specifying the decision framework at the subsystem-level. Based on this framework, the subsystem design is selected.

This optimization process can be formulated mathematically as:

Architecture level optimization:

$$\underset{\gamma_{motor},\gamma_{gen}}{\underline{Min}} \quad Wtot\begin{pmatrix} W_{motor}\left(\overline{\gamma}_{motor} Cap_{motor}\right), Loss_{motor}\left(\overline{\gamma}_{motor} Cap_{motor}\right),... \\ W_{gen}\left(\overline{\gamma}_{gen}, Cap_{motor}\right), Loss_{gen}\left(\overline{\gamma}_{gen} Cap_{gen}\right) \end{pmatrix}$$

Subjected to:

- $Cap_{motor} = MP$ **(47)**
- $Cap_{gen} = EP$
- $\{\gamma_{1-motor}, \gamma_{1-gen}\} \in [0,1]^2$
- $\gamma_{1-motor} + \gamma_{2-motor} = 1 \text{ and } \gamma_{1-gen} + \gamma_{2-gen} = 1$

Subsystem level optimization:

$$\underset{X}{Min} \quad \gamma_1 \frac{Loss(X)}{Loss|_{ideal}} + \gamma_2 \frac{W(X)}{W|_{ideal}} \tag{48}$$

Subjected to:

- $Cap = \left(a_1 \times Loss|_{\min} - b_1\right)^{p_1} \times \left(a_2 \times W|_{\min} - b_2 \times Cap\right)^{p_2}$

5.3.4.1.3 Pareto Front used in Experiments

In this experiment three types of fronts will be considered. The first type will be referred to as the "convex" problem because both the motor and generator fronts are strictly convex. The second type corresponds to a "hybrid" problem where the front of the motor is strictly convex, but where the generator's front is non-convex. The third problem is based on a double non-convex problem and is referred to as the "non-convex" problem.

Table 24: Front parameters for experiment 3.2 and 3.3

Convex problem

Motor Front				Tech settings Gen			
Front 1	a:	0.8	1	Front 1	a:	0.8	1
	b:	0.1	1.5		b:	0.1	1.5
	p:	0.5	1		p:	0.5	1

Hybrid problem

Motor Front				Generator front			
Front 1	a:	0.8	1	Front 1	a:	0.8	2
	b:	0.1	1.5		b:	0.2	1.5
	p:	0.5	1		p:	0.7	1
				Front 2	a:	0.8	0.5
					b:	0.1	3
					p:	0.2	1

Non-convex problem

Motor Front				Generator front			
Front 1	a:	0.8	1	Front 1	a:	0.8	2
	b:	0.2	1		b:	0.2	1.5
	p:	0.3	1		p:	0.7	1
Front 2	a:	0.8	0.5	Front 2	a:	0.8	0.5
	b:	0.1	5		b:	0.1	3
	p:	0.2	1		p:	0.2	1

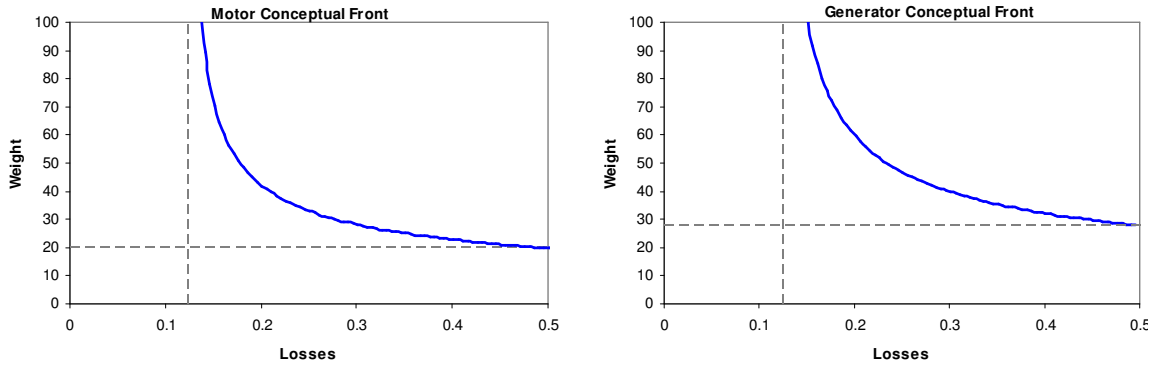


Figure 152: Convex problem

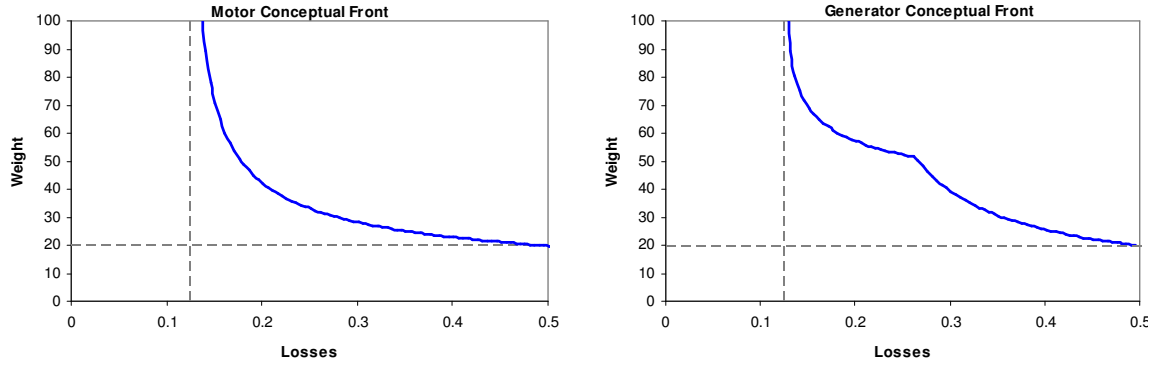


Figure 153: Hybrid problem

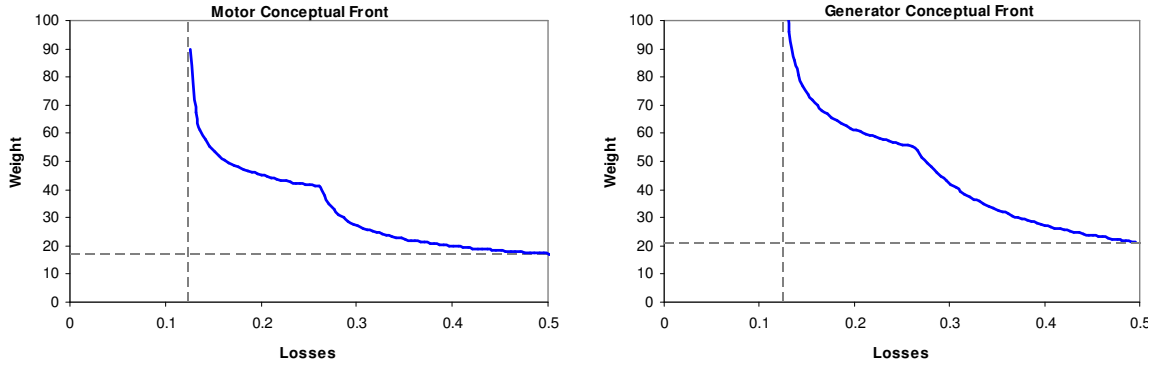


Figure 154: Double non-convex problem

5.3.4.2 Observation of Topologies Implied by the Optimization Processes

5.3.4.2.1 Observations on the All-in-One Optimization Process

In order to observe the how an All-in-One optimization process (AiO) resolves the design problem, we shall start by observing the topology of the space it implies. This space is parameterized by the two variables used in this optimization process: X_{motor} and X_{gen}. In order to have a more meaningful idea of what these variables imply, we shall consider them via the efficiency figure they represent. The equation linking X to the efficiency is presented below. This equation is based on equation (41) presented earlier.

$$Eff(X) = 1 - \left[Loss\big|_{ideal} + X \times \left(Loss\big|_{\max} - Loss\big|_{ideal} \right) \right] \quad \textbf{(41)}$$

Using this equation the optimization space of the All-in-One optimizer was explored for the three design test problems presented earlier. These spaces were defined based on the iterative evaluation of the architecture overall Figure of Merit for all combinations of X_{motor} and X_{gen} between zero and 1.

5.3.4.2.2 Overview of the topology

The topology of the Coordinated Optimization process (CoOp) is presented using $\gamma_{1\text{-motor}}$ and $\gamma_{1\text{-gen}}$. The priority factors are varied between 0 and 1. Both $\gamma_{1\text{-motor}}$ and $\gamma_{1\text{-gen}}$ correspond to the priority factors dedicated to the efficiency term in the subsystem objective function. The priority factors dedicated to the same subsystem are expected to sum up to one. When $\gamma_{1\text{-motor}}$ is equal to one, it means that the motor design is only optimized on efficiency (with no regards to its weight). Vice versa, if $\gamma_{1\text{-motor}}$ is equal to zero, the motor is optimized purely on weight.

In order to get a feel for the topology implied by each optimization approach, contour plots were created. Figure 155 presents an overview of the topology associated with the three design problems described earlier. A more detailed discussion of this figure is provided in the next paragraph.

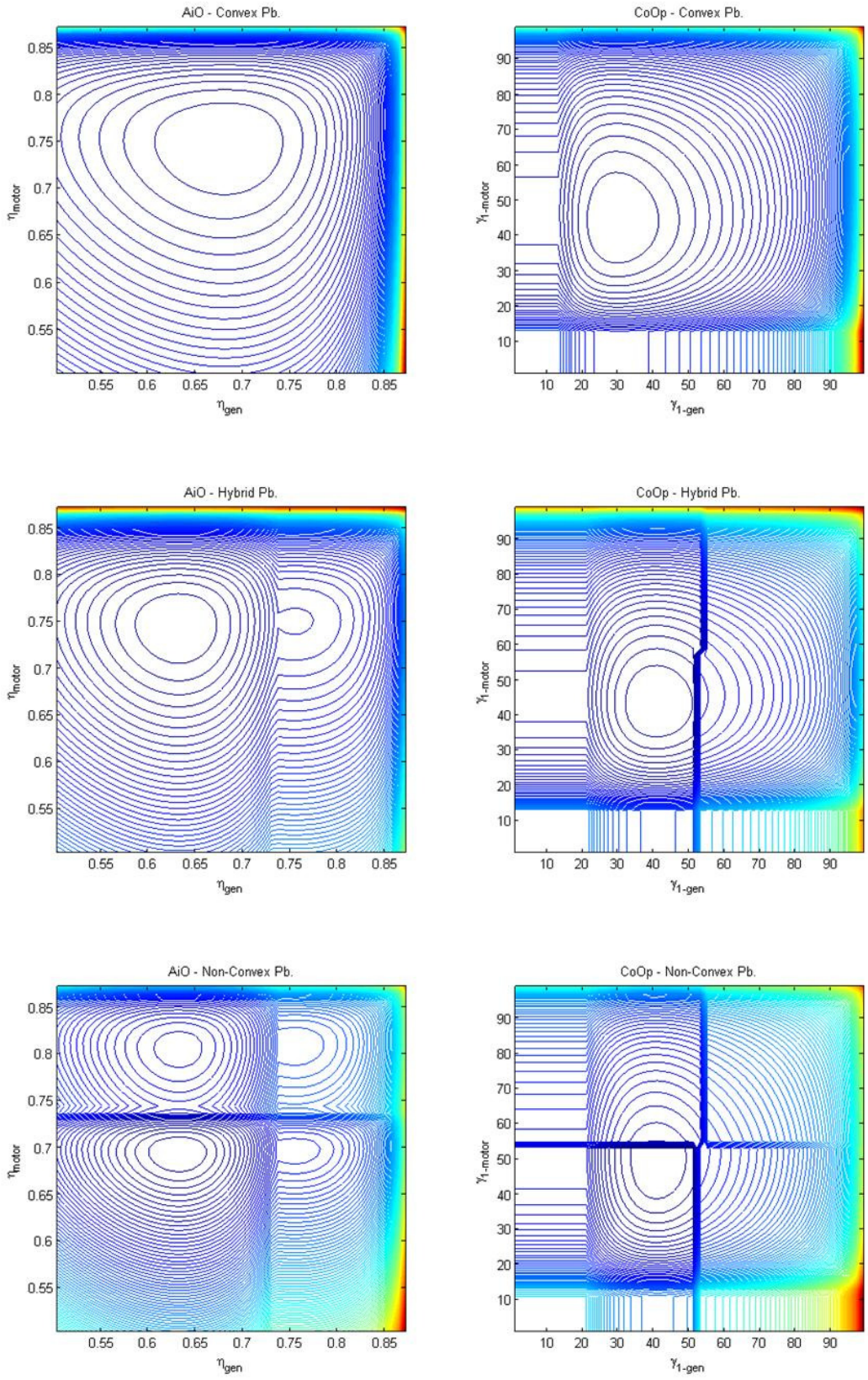


Figure 155: Overview of topologies

5.3.4.2.3 Analyzing the Shapes of Topologies

In order to understand the topologies shown in Figure 155, one has to consider the location of the selected motor and generator designs on their respective Pareto fronts. Figure 156 presents the designs corresponding to two design points with the CoOp approach. The first design point corresponds to the optimal design. The attributes of the motor and generators associated with this design are presented in the upper right corner of the figure. The second design labeled "other" presents a different solutions with its associated attributes presented on the bottom right corner of Figure 156. The design of the motor and generators are represented on their respective Pareto fronts. The designs boxed in red correspond to the optimal design (top and left Pareto fronts) and the greened boxes to the "other" (right and bottom fronts).

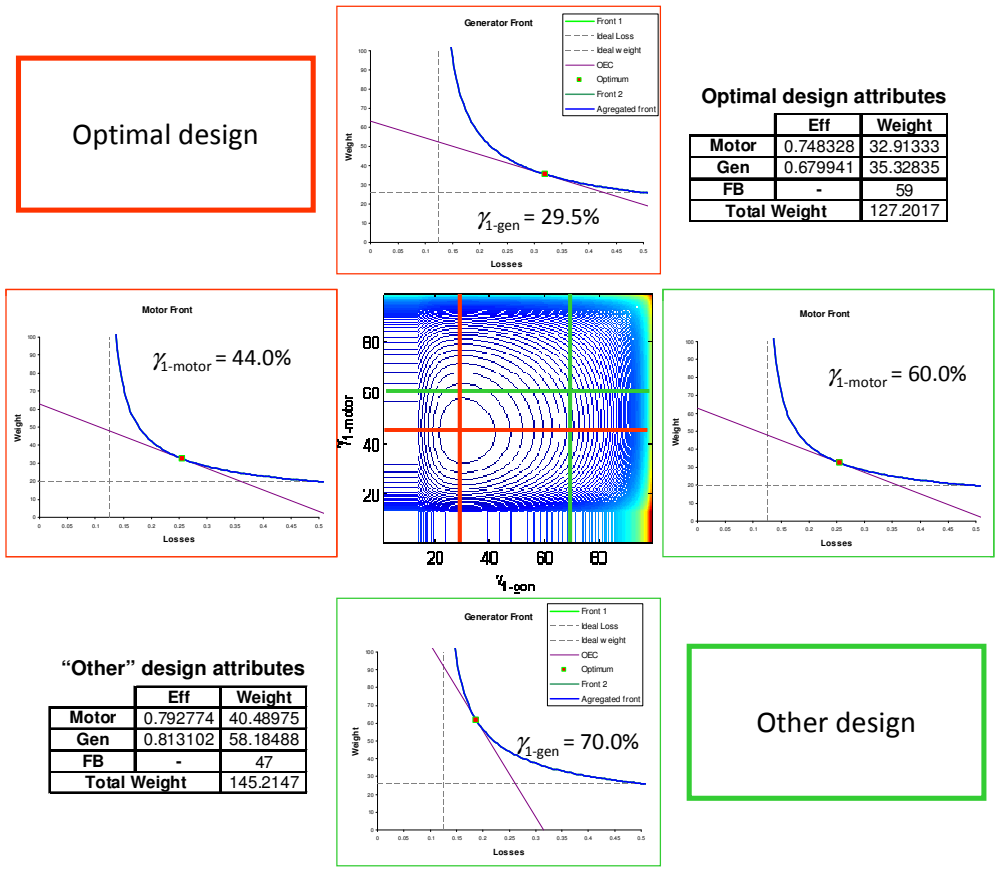


Figure 156: Design details associated with topology

If we consider the topology presented by the CoOp approach we can notice that for low values in $\gamma_{1\text{-motor}}$ and $\gamma_{1\text{-gen}}$ the architecture becomes insensitive to their change. This is the expression of a limiting rule which placed a limit on the worst efficiency design point that can be selected on the front. Therefore, as the value of $\gamma_{1\text{-motor}}$ and $\gamma_{1\text{-gen}}$ are decreased below a certain threshold the design of the subsystem remains fixed (see Figure 157 for a graphical illustration).

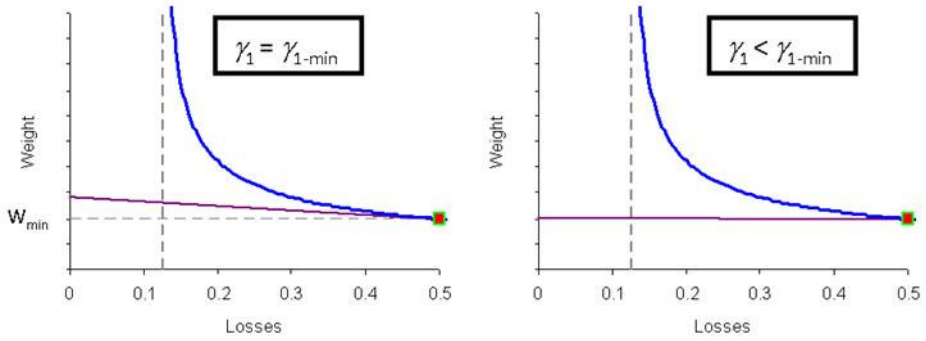


Figure 157: Illustration of the Pareto threshold

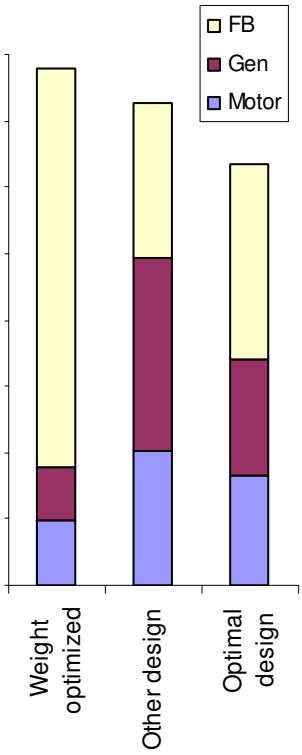


Figure 158: Weight breakdown

In order to better understand the topology illustrated in Figure 156, a weight breakdown is presented for three specific priority factor sets. The first two sets correspond to the optimal designs presented in Figure 156 (the optimal design along with the design labeled “other”). The third set corresponds to the point located in the lower left corner of Figure 156 and which represents the pure weight optimization design. For each of these optimization strategies, we can observe the contribution to overall weight associated with each subsystem.

Based on this illustration, we can see that despite the intent to directly optimize the weight by minimizing directly the weight of the generator and the motor, the weight of the architecture was suboptimal due to the resulting fuel burn. By compromising on the subsystem weight an improved integrated system performance could be achieved as illustrated by the “other” and optimal designs.

A comparison between the AiO and the CoOp optimization is available in Figure 159. The line on this figure presents the correspondence between locations in the two spaces (the locations pointing to the same design). What is important to note is that the redline connecting both optimal locations in the space correspond to the same design. For the convex problem both problem returned an optimal design with an efficiency of 68.0% for the generator and 74.8% for the motor. As we will see later-on the same conclusion is reached for the hybrid and non-convex case. This allows us to conclude that if the CoOp approach will optimize the architecture in a different space, it still yields the same optimum. This observation provides evidence supporting *Hypothesis 3.2.*

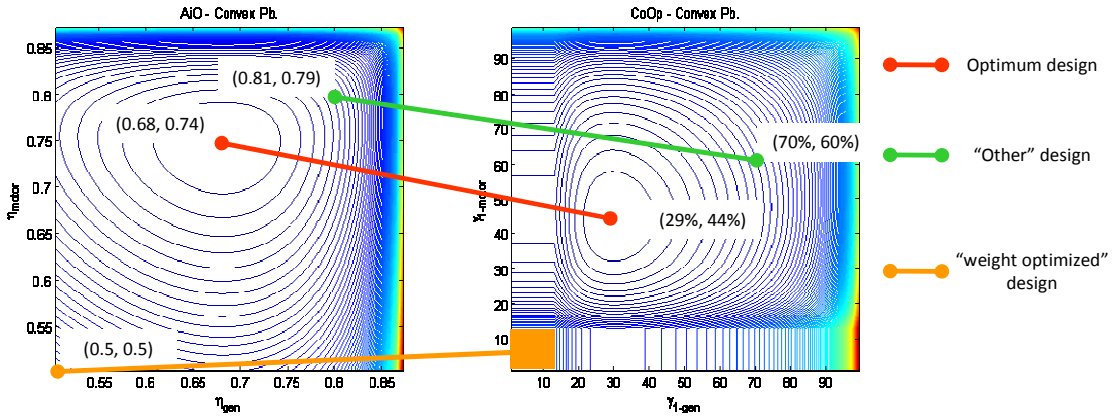


Figure 159: Comparison of convex problem topologies

5.3.4.2.4 Discontinuities with Non-Convex Pareto Fronts

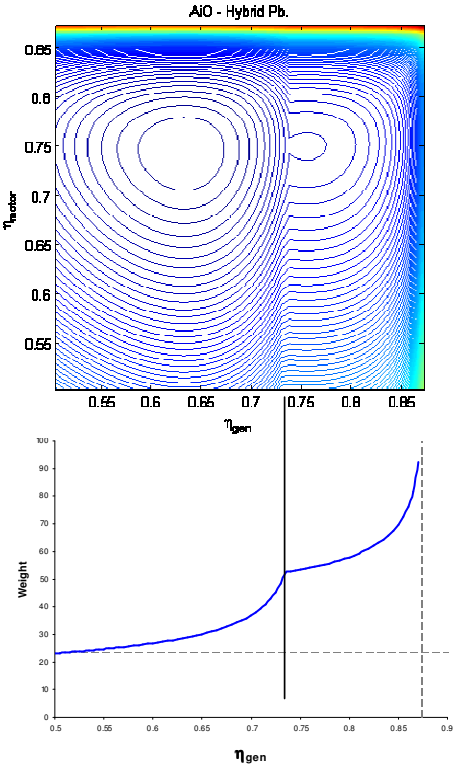


Figure 160: Pareto discontinuity for AiO approach

The discontinuity in the Pareto front implies a distinct inflection in the topology implied by the AiO approach. Because of this inflection, the topology of the AiO optimizer is no longer convex (with one local optimum). This inflection can be better

understood by considering the cavity inside the Pareto front (the region around the discontinuity in the Pareto front). In Figure 160 we can see that as we move toward the center of this region, the degradation in one of the attributes will exceed the improvement in the other. At the system-level this poor compromise is expressed by general degradation of the overall figure of merit. This local degradation creates a "ridge" in the overall topology which splits the optimization into 2 areas, each area containing a local optimum.

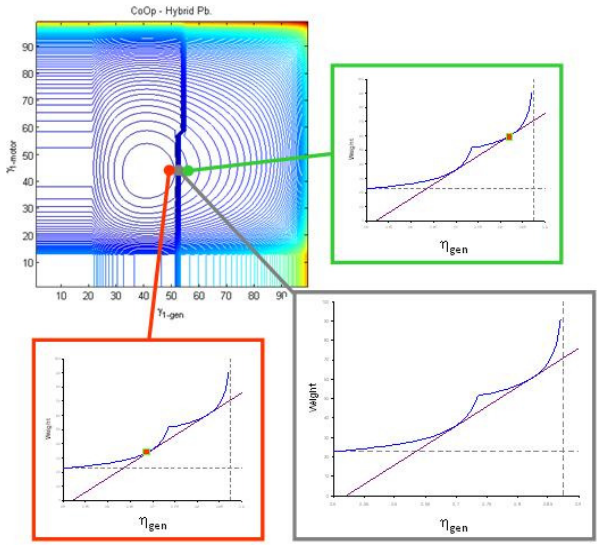


Figure 161: Pareto discontinuity for CoOp approach

Given the problem formulation implied by the CoOp approach, the effect of the Pareto discontinuity on the topology is significantly different from the AiO. The subsystem design is selected by the optimization of the subsystem objective function. This approach avoids suboptimal subsystems. In this situation, the designs located in the Pareto cavity are considered suboptimal and therefore are avoided altogether (see Figure 161). Note that Das et al. have also noted this property in a different context [79].But when the subsystem optimizer switches from one side of the front to the other, a slight change in the priority factors will imply a large variation in subsystem design. This variation implies a discontinuous change in the system-level objective. This discontinuity is clearly visible in Figure 161 which represents the topology for the hybrid problem. The

discontinuity is located at values of $\gamma_{1\text{-gen}}$ near 50%. It is important to observe that the topology implied by the CoOp approach is discontinuous, but with a single local optimum. This situation greatly simplifies the optimization process at the architecture level. This simplification is even more obvious by observing the topologies associated with the non-convex problem where the optimal solutions for both generator and motor lay on non-convex Pareto fronts. In the associated topologies we can see that the AiO' includes four local optimum while the CoOp' includes a single global optimum.

5.3.4.2.5 Risk of Suboptimal Solutions Associated with Improper Subsystem Objective Functions

In the previous paragraph, we observed that the CoOp approach avoids using points in the cavity of the Pareto front based on the subsystem objective function. It is important to note that although not considering this region simplifies the architecture-level optimization process, it creates a "blind spot" in the optimization process. As shown in Figure 161, a slight change in the priority factors will lead to an abrupt change in the selected subsystem solution. This abrupt change results from overlooking solutions located in the concave zone on the front.

The subsystem objective is only a means to represent the architecture-level objective. In other words the subsystem objective function is an approximation of the overall objective function. If the approximate form does not capture all the effects necessary to the representation of the architecture-level objective, we may end up in situations where the architecture optimal solution is overlooked (i.e. it is located in the "blind spot"). In the non-convex problem considered in this thesis it was not the case. The blind spots in this experiment are presented in Figure 162.

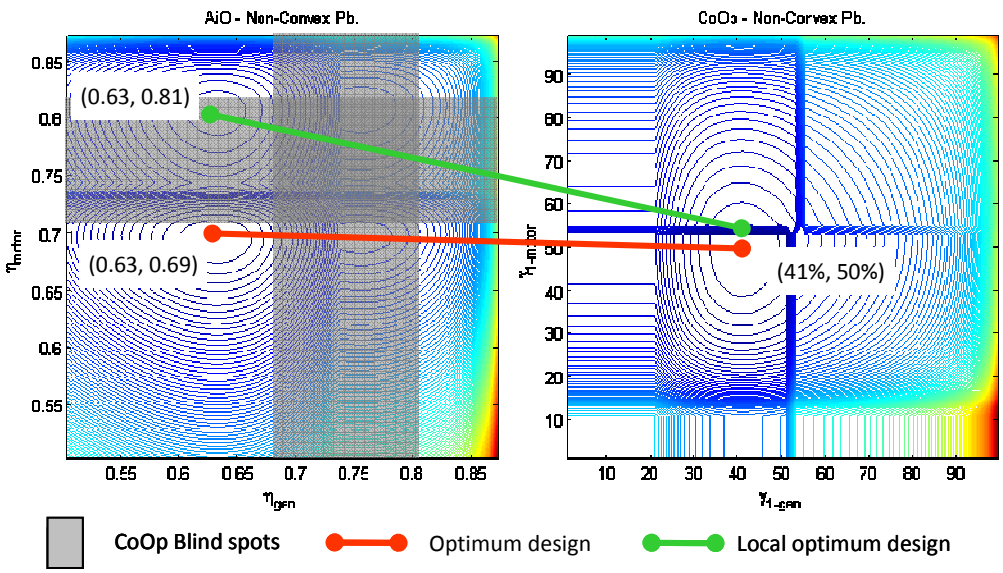


Figure 162: Blind spots of the CoOp in the non-convex problem

In order to avoid situations where the subsystem optimizer may overlook architecture optimal solutions, it is necessary to have a general understanding of the shape of the architecture objective function in the subsystem attribute space. In order to illustrate this idea, Figure 163 was constructed. It presents two types of architecture-objective functions. The first, described as the "convex" architecture objective will never require solutions within the front cavity, while the second (or the "concave" architecture objective) will. Based on this observation, we can conclude that two aspects must be carefully observed when using CoOp. The first is the shape of the Pareto front. If this front is non-convex, it is important to ensure that the architecture objective is not concave. If both the front and the architecture objective functions are non-convex, linear subsystem objective functions will overlook architecture optimal solution. In this situation it is necessary to identify the proper form of the architecture objective function with respect to the subsystem attributes. Based on this form, a parameterized objective function can be formulated to more closely match the objective function.

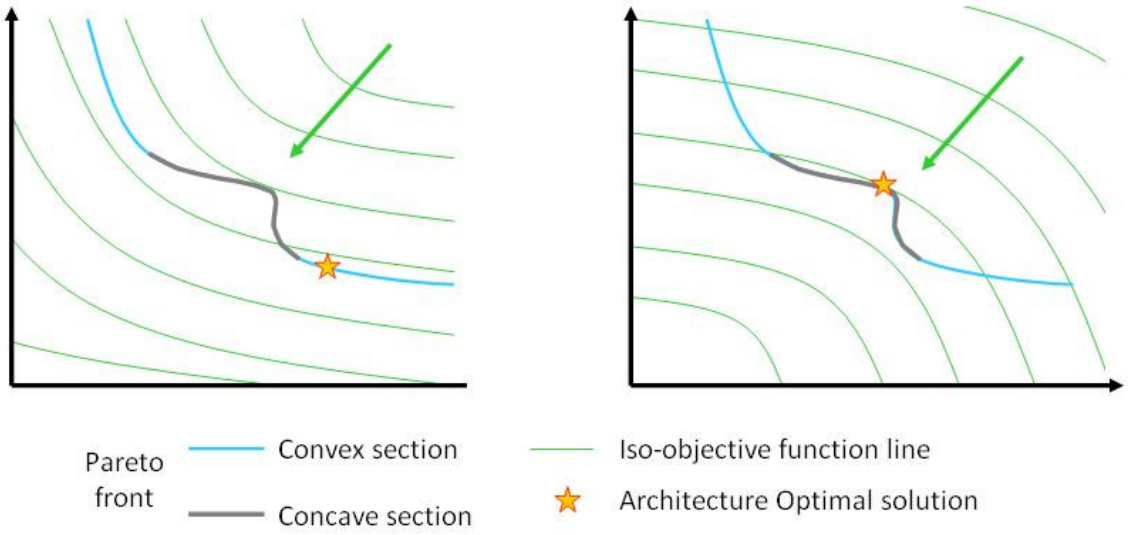


Figure 163: Architecture optimal solution in concave section

5.3.4.3 Conclusions

Under several points of view, the All-in-One optimization (AiO) approach offers an intuitive approach to the optimization of the architecture. Optimization variables have a direct physical meaning and the trade-offs they imply can directly be understood. On the other side, the Coordinated Optimization (CoOp) approach is based on an indirect vision of the physical trades at the subsystem level. Instead of optimizing a physical parameters like a geometrical parameter, the CoOp approach will trade preferences between the minimization of the weight or the energy consumption of the subsystem. Although these parameters do not carry a direct meaning in terms of a physical solution, they do carry an important meaning from a subsystem design perspective. Instead of pointing to a solution, the CoOp approach formulates a design problem which can be then optimized at the subsystem-level. Now instead of dealing with the subsystem design trade-offs, the optimizer at the architecture-level can focus on system-level trades. As presented earlier, this approach allowed for the simplification of the architecture-level topology compared to the AiO approach. This simplification was unambiguously illustrated by the fact that the CoOp approach implied a single optimal point for the

hybrid and non-convex problems, while the AiO topology was facing multiple local optimal solutions.

In this experiment some limitations were observed concerning the capability of the CoOp approach to identify the architecture optimum set of subsystem solutions. The disadvantage of pre-optimizing the subsystem implies that some solutions will not be considered. If the objective function at the subsystem-level is not properly formulated, the architecture optimum may be hindered by the subsystem-level optimizer.

The proposed CoOp approach delegates some part of the architecture optimization to subsystem-level optimizers. This approach allows for a subsystem-centric sizing approach necessary to the modular construction of the architecture model. Experiment 3.2 has shown that the CoOp approach:

- Provides the ability to converge to an architecture optimum
- Simplifies the architecture level optimization process

But these advantages require that an appropriate objective function form was formulated at the subsystem level. This formulation requires some insight on the impact of subsystem attributes on the global architecture.

Hence, we may conclude that *Hypothesis 3.2* was validated. The CoOp approach offerst clear advantages in term of architecture optimization and developments. But in order to work properly, it requires basing the subsystem-level optimizer on the proper objective function form.

5.3.5 **Experiment 3.3: Comparison of Subsystem Design Problem Formulation**

Experiment 3.2 has observed the effects of the Coordinated Optimization approach on the conceptual activities. Now experiment 3.3 will analyze the effect of the proposed approach on the subsequent development phase. In chapter 2, we observed that models used in conceptual design always include some degree of imprecision.

Nevertheless, it is based on these imprecise models, that decisions are made in conceptual stages where the subsystem design problem is defined. This experiment will analyze the effectiveness in dealing with the uncertainty of several subsystem problem formulations. One of these formulations results from the CoOp approach and will be compared with those defined by traditional optimization methods applied to conceptual design.

5.3.5.1 Presentation of the test considered by experiment 3.3

The objective of experiment 3.3 is to test Hypothesis 3.4 (*Formulating subsystem design problem as an optimization problem enables better subsystem developments*). This experiment will simulate subsystem developments. Subsystem developments will be described by optimization processes. In order to represent the variability inherent to developments, the optimization processes will be based on models which deviates slightly from the conceptual models. This deviation, generated by random variables applied to the conceptual model, will simulate this variability and the lack of accuracy of the conceptual model it implies.

In order to represent Clustered Concurrent developments, we shall assume that it is not possible to optimize the subsystems based on an architecture level objective. Instead the optimizers simulating subsystem developments will use the information previously defined in conceptual design. Therefore, the starting point of this experiment is the conclusion of the optimization processes presented in experiment 3.2. In the previous experiment, the All-in-One and CoOp optimization processes have both identified optimal subsystem solutions. In addition to the identification of the solutions, the CoOp has also identified objective functions for each subsystem design problem. Hence, based on their conclusion, each conceptual optimization approach implied different subsystem design problems. The All-in-One has implicitly defined target based subsystem design problems while the CoOp has formulated objective-based design problems.

This experiment will test the two formulations of the subsystem design problem by comparing the performance of the architectures they induce. These approaches are described in the following paragraph.

5.3.5.1.1 Introduction of Subsystem Design Problem Formulations

This experiment compares subsystem development based on two formulations. The first formulation will be referred to as the target-based design problem, the second as the objective-based design problem.

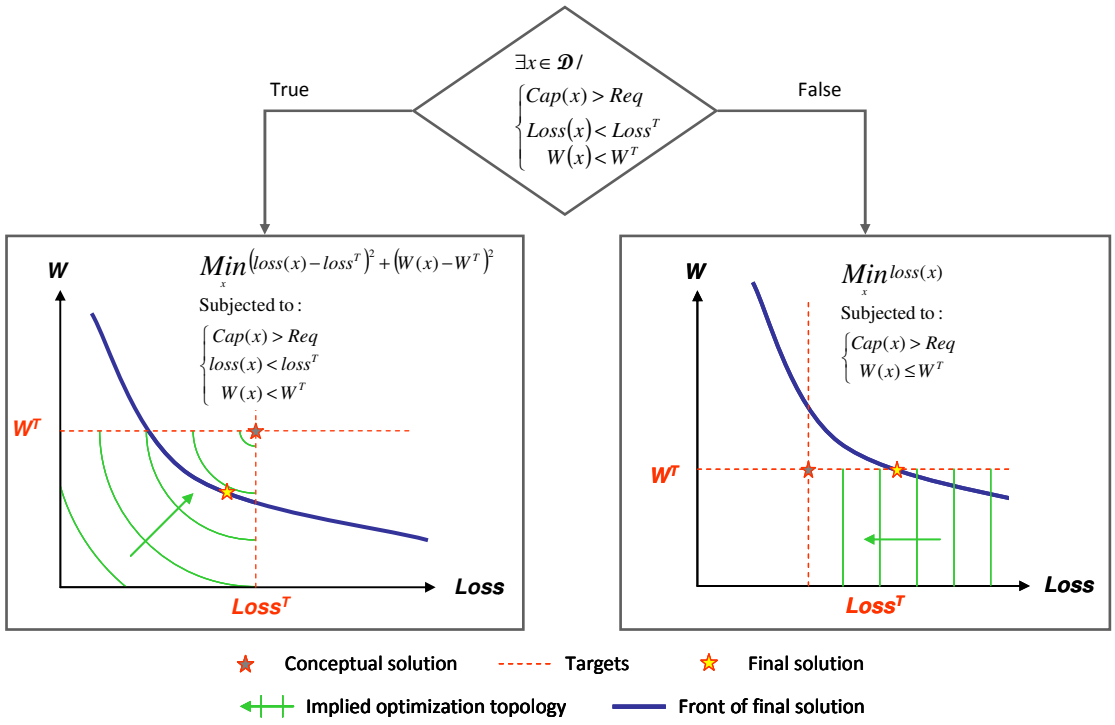


Figure 164: Target-based design problem

The "target-based" design problem is an improved target matching problem. In this context, the selection process will follow the logic presented in Figure 164. If the conceptual design model was pessimistic (i.e. weight and loss were underestimated), the designer will try to choose the point on the front which is in closest proximity with the target specified in conceptual design. This situation is illustrated by the optimization problem on the left part of Figure 164 and will be referred to as the "proximity-to-the-

target" mode. If the conceptual solution was optimistic (i.e. the weight and loss target can not be achieved), the designer will try to meet al least one of the two targets. In this experiment, the designer will try to meet the weight target by relaxing the loss target. Consequently, the final design will correspond to the solution with the smallest loss possible which can meet the weight target. This situation which is the nightmare of all system designers will be referred to as the "save-at-least-one-target" mode.

The other approach considered is the objective based design problem. The problem formulation corresponds to what was proposed earlier as the subsystem sizing approach – equation (48) reproduced below.

$$\underset{X}{Min}\ \gamma_1 \frac{Loss(X)}{Loss|_{ideal}} + \gamma_2 \frac{W(X)}{W|_{ideal}} \tag{48}$$

Subjected to:

- $Cap = \left(a_1 \times Loss|_{\min} - b_1\right)^{p_1} \times \left(a_2 \times W|_{\min} - b_2 \times Cap\right)^{p_2}$

A graphical representation of the design problem is provided in Figure 165. It is important to note that this type of problem definition does not include any constraints based on the conceptual solution. The only information kept from conceptual design is the optimal objective function (i.e. the optimal priority factors)..

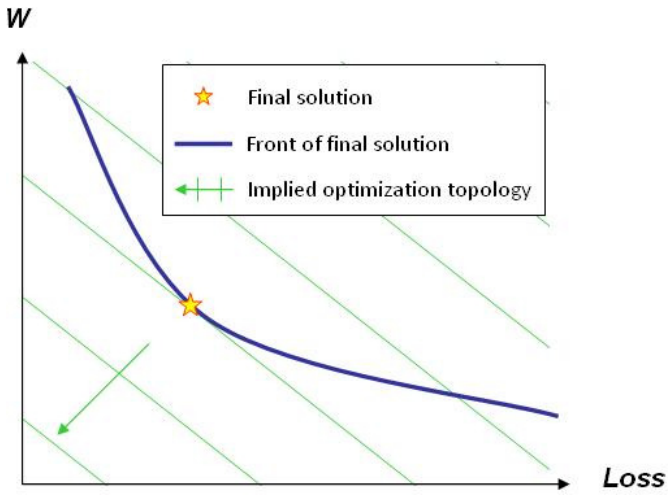


Figure 165: Objective-based design problem

5.3.5.1.2 Representation of Conceptual Model Inaccuracy

In this experiment the model for the motor and the generator are represented by the parameterized Pareto front presented in the constraint of equation (48). This Pareto front is characterized by six parameters (a_1, b_1, p_1, a_2, b_2, and p_2.) which define its shape. We shall assume that the final solutions can be represented by a Pareto front also defined by the constraint of equation (48) but with different values for parameters a_1, b_1, p_1, a_2, b_2, and p_2. The parameters describing the final solution front are identified by the * symbol. These parameters are defined probabilistically using the following normal distributions:

$$a^* = N\left(a\ ,\sigma_a\right) \quad \textbf{(49)}$$

$$b^* = N\left(b\ ,\sigma_b\right) \quad \textbf{(50)}$$

$$p^* = N\left(p\ ,\sigma_p\right) \quad \textbf{(51)}$$

In these equations the σ terms express the deviation that is expected on the front parameters. The more uncertain the parameters are (i.e. the more immature the technologies considered in conceptual design), the higher their value. In order to test *Hypothesis 3.4* in different contexts of uncertainty, several tests will be performed in this experiment. The first test will imply developments with low uncertainty, the second with higher uncertainty. The normalized values corresponding to each σ are presented in the following table.

Table 25: Standard deviation applied in experiment 3.3

Sigma	Low uncertainty	High uncertainty
σ_a / a	3%	6%
σ_b / b	5%	10%
σ_p / p	2%	4%

5.3.5.2 Comparison of Simulated Developments

5.3.5.2.1 Implementation of the Experiment

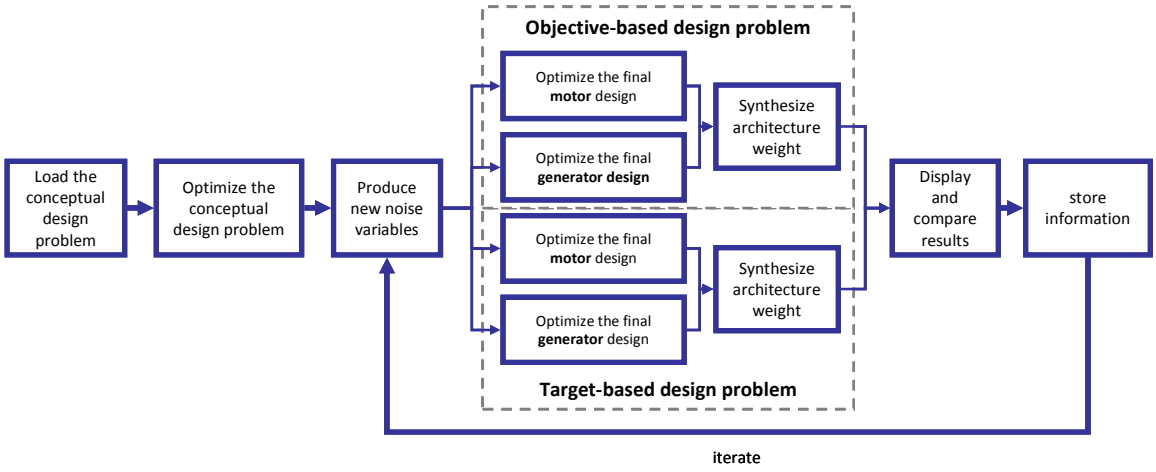


Figure 166: Development simulation process

In order to automate the simulation of subsystem developments, a simple Excel tool was implemented. This tool allows for the optimization of the conceptual design problem (identification of the subsystem targets and priority factors), and the iterative simulation of subsystem developments based on different deviation scenarios (using the distribution presented in Table 25). The process of execution of the tool is presented in Figure 166. In order to capture the probabilistic nature of the problem, the experiment will compare iteratively the performance of the architectures obtained based on each approach. Each run will generate a different scenario (i.e. a different Pareto front for the final subsystem solutions) using the distribution specified in equations (49), (50) and (51). Conclusion will be drawn probabilistically based on the performance of each approach over the population of scenarios generated.

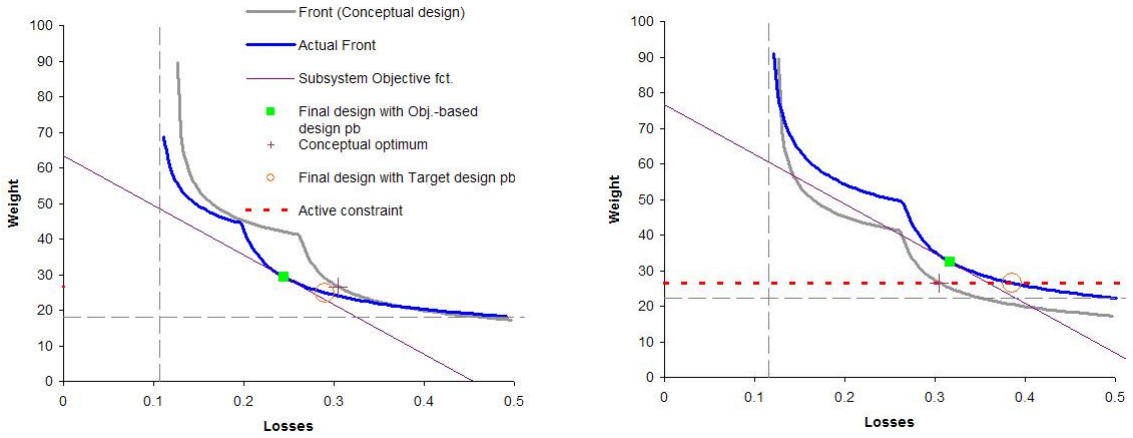
- the subsystem providing the function (the source)
- the subsystem receiving the function (the load)
- the type of function relating the two elements

5.3.5.2.2 Observation of the Final Subsystem Selection

For each run the tool provides a visualization of:

- The conceptual Pareto front with the optimal solution
- The actual Pareto front associated with the final solution
- The final solution designated by each design problem

This information is provided for both the motor and the generator. An example of the information provided is provided by the snapshots presented in the following figure.



(without active constraints) (with active constraints)

Figure 167: Example simulated developments

The plot on the left shows the solution associated with a situation where the conceptual model was too conservative (i.e. the attributes predicted were not as good as those that could eventually be realized). This situation can be recognized from the fact that most of the grey front (conceptual Pareto front) is located above and to the right of the actual front (presented in blue). Using the two design problems stated earlier, two distinctly different final solutions were selected. The first solution represented by the green dot was selected based on the objective-based design problem while the second solution associated with the target based design problem, selected the point located at the red circle. We can see that in this situation, the objective-based design problem has steered the subsystem toward a solution which slightly compromised the weight estimation but obtained a major improvement in efficiency in return. The target-based

approach was operating in its “proximity-to-the-target” mode and provided a solution which satisfied both targets but without benefiting much from the favorable outcome in the realization of the subsystem.

The right hand side plot presents a situation where conceptual estimations were overly-optimistic (i.e. the attributes predicted could never be met by the actual developments). In this situation the target-based design problem formulation will push the designer to be in the “save-at-least-one-target” mode. In order to at least meet the weight target, the target-based design problem will vastly compromise the efficiency target. On the other hand the objective-based approach will converge to the point optimizing the objective function.

5.3.5.2.3 Probabilistic Comparison of the Target-Based and Objective-Based Approaches

The experiment was composed of six tests. The tests were performed for the three subsystem model shapes (convex, hybrid and non-convex). For each problem, two tests were performed. The first test corresponded to a situation where the variation between the actual performance of subsystems and their predicted value in conceptual design was low. The second corresponds to a high variation.

Each of these six tests was performed over a population of 1000 samples. The values for the noise associated with each sample were defined using the distributions specified in equations (49), (50) and (51) in association with the values presented in Table 25. The low variance tests were designated in the tests by the term "class 1 error", while the high variance tests were qualified as "class 2 error". The results of the tests are presented in Figure 168 and Figure 169 and exampled in the following pages.

The impact of using objective-based subsystem design problems instead of target based can be observed from several perspective. The most noticeable impact is the probability to realize a better architecture using objective-based design problems rather than target-based. The probabilities associated with the six tests are listed in Table 26. This table provides evidence supporting the validity of *Hypothesis 3.4* in the context of the problem considered in this experiment.

Table 26: Probability of improvements with Objective-Based design formulation

Type of fronts	Low conceptual uncertainty (class 1 error)	High conceptual uncertainty (class 2 error)
Convex	100%	99.6%
Hybrid	98.5%	97.6%
Non-convex	97.2%	95.67%

The fact that all figures listed in Table 26 are well above 50% means that the methodological approach had an impact on the physical outcome (i.e. the performance of the architecture). The results in this table demonstrate unequivocally that better outcomes (at the architectural level) are achieved by developing a subsystem using an objective function. As hypothesized earlier in this dissertation, this approach provides the subsystem designer with the information and design freedom necessary to either exceed expectations set in conceptual design (for situations with a favorable development outcomes) or limit the degradation of architectural performance (in situations where conceptual objectives were unrealistic).

The amplitude of the improvements is difficult to evaluate. Figure 169 presents the spread of the improvements enabled by the objective-based approach from the target-based approach. This spread changes with the form of the problem (convex, vs. hybrid, etc…) and the amplitude of the conceptual uncertainty.

The improvement is clearly noticeable, however, when we consider the probability of success in achieving a specific level in the architecture objective. The Probabilities of Success (PoS) are shown in Figure 168. They express the probability of reaching a specific value in the architecture level objective. We can see that the blue line showing the probability associated with the objective-based approach is above and to the left of the red lines presenting the probabilities associated with the target-based approach. Therefore, regardless of the value that we would like to reach at the architecture level, this experiment demonstrates that including an objective function to define subsystem requirements will increase the probability of success at the architecture level. The amplitude of this increase depends on the nature of the technical problem and the level of uncertainty. But the improvement in probability of success can go up to values exceeding 25% which is considerable!

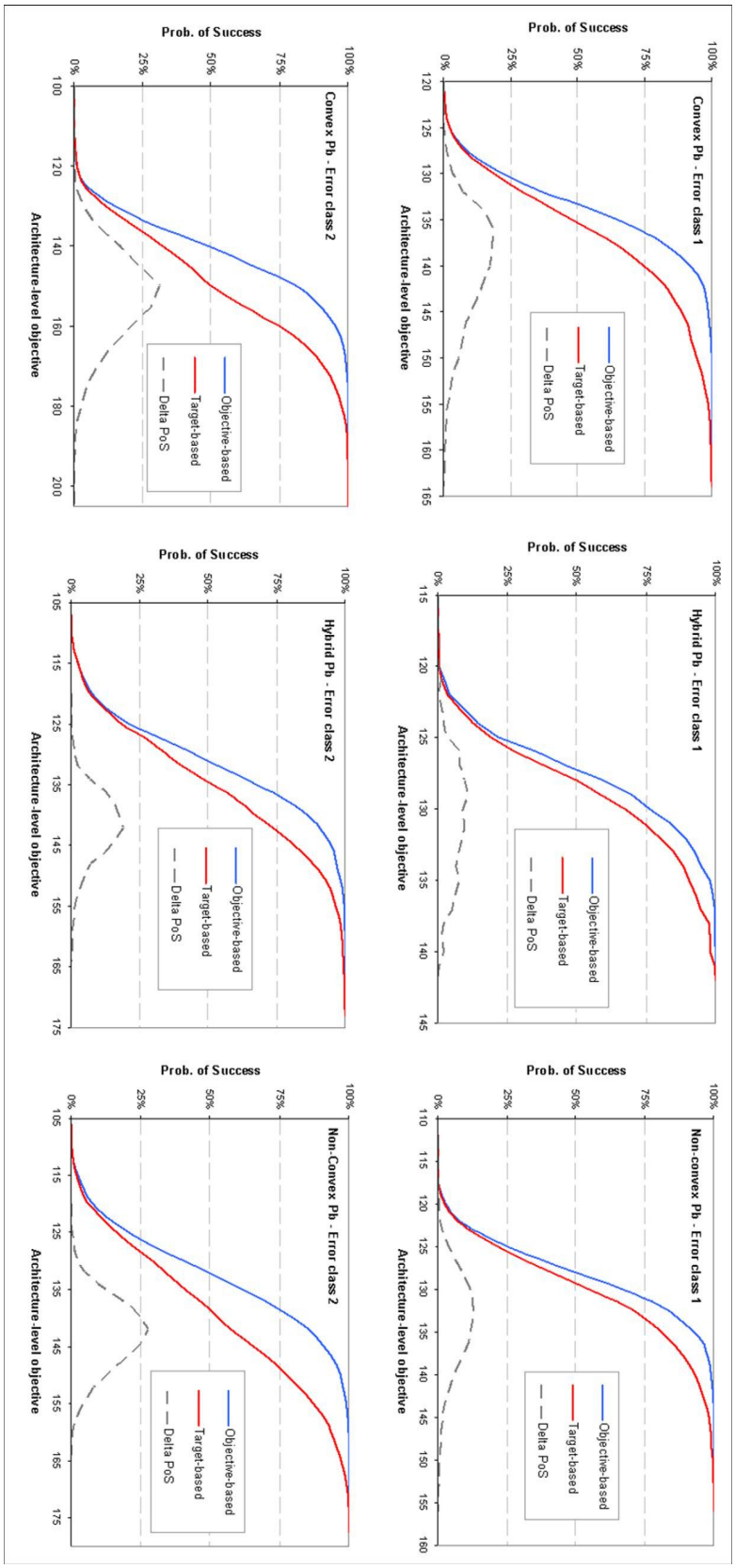


Figure 168: Probability of success in achieving an architecture objective

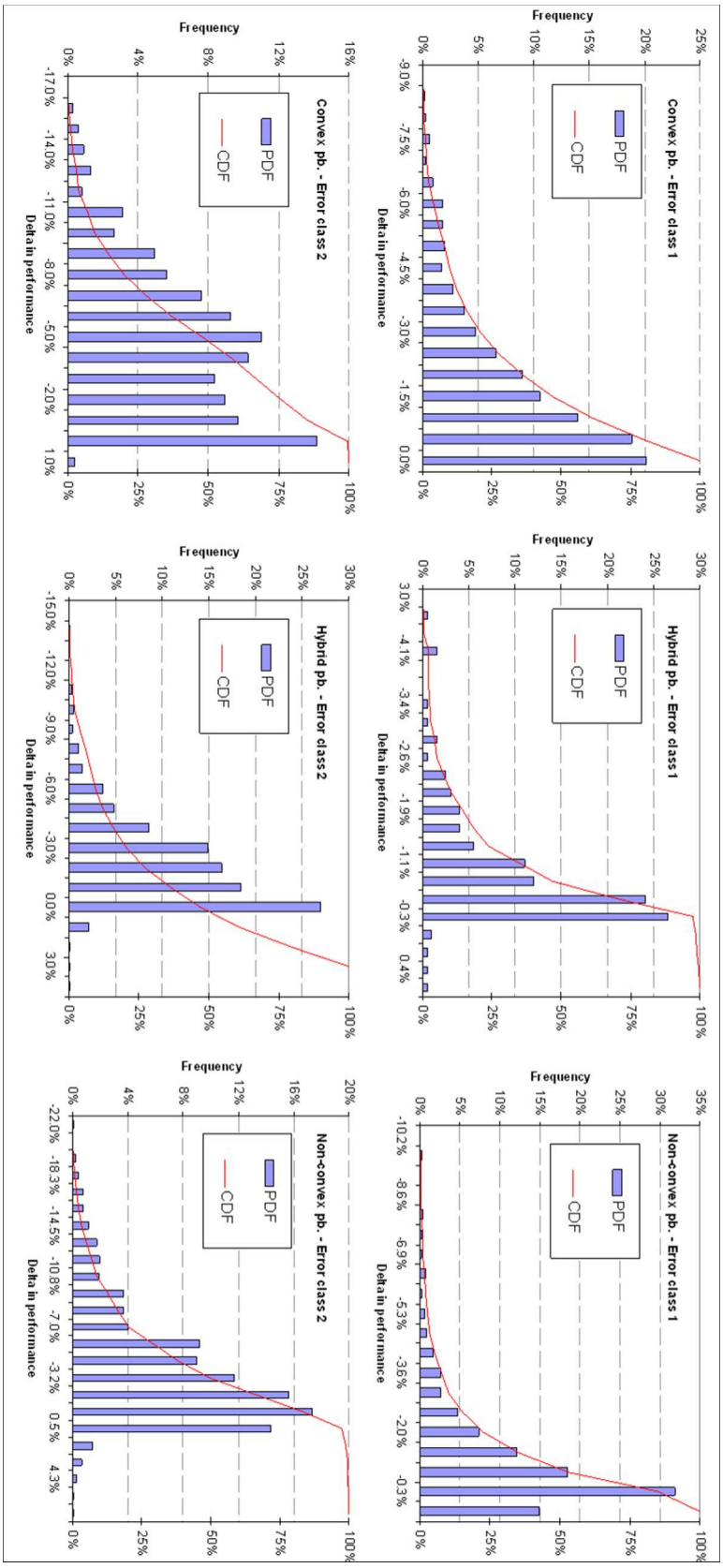


Figure 169: Spread of improvements resulting from objective-based design problems

5.3.5.3 Robust requirement Formulation

Formulating subsystem requirements based on an objective function formulates subsystem directives in a more robust fashion than the target based. If this important observation led to the validation of *Hypothesis 3.4:* it is also important to note that this technique allows to address uncertainty in conceptual design models. This uncertainty is unavoidable and must be taken into account in the formulation of requirements for future developments.

It is interesting to observe that objective-based requirements formulate more robust directives without needing to perform computationally costly Monte Carlo simulations. In the experiment above the Monte Carlo method was only used to demonstrate the ability of the technique to produce robust solution. This robustness results from the fact that with the formulated subsystem objectives, the subsystem designer is capable to implement a recourse strategy (if conceptual design estimations were optimistic) or to further optimize their subsystem (if conceptual design estimations were pessimistic). Therefore instead of attempting to find a robust conceptual design, this approach addresses uncertainty by providing flexible requirement to subsystem development. This flexibility eventually enables the more robust developments presented in experiment 3.3.

5.3.6 **Conclusions on Coordinated Optimization Experiments**

5.3.6.1 Philosophical Overview

In this dissertation, the Coordinated Optimization (CoOp) approach was formulated with the objective to:

- Facilitate and improve the quality of the sizing process of subsystems
- To guide subsequent development in a context of clustered concurrent engineering process.

This method was based on three hypotheses:

Hypothesis 3.2: Coordinated optimization will yield the same optimized solution as a direct optimization approach.

Hypothesis 3.3: The optimal priority factors are proportional to the gradient of the system-level objective

Hypothesis 3.4: Formulating subsystem design problem as an optimization problem enable better subsystem developments

In experiment 3.2, we have observed that the CoOp approach was comparable to a classical optimization approach. Thanks to its subsystem-level optimization process, this approach even has the ability to simplify the architecture-level optimizer topology by readily eliminating suboptimal subsystem solutions. This experiment was able to demonstrate a successful realization of *Hypothesis 3.2*. Through this demonstration we were able to show that *Hypothesis 3.2* addresses *Research Question 4.2*.

In experiment 3.1 we have shown that, at the optimal solution, the priority parameters define an objective function for the subsystem which is locally equivalent to the architecture level optimization. This observation validates the keystone *Hypothesis 3.3* and demonstrated that the Coordinated Optimization method (originally designed to facilitate the sizing process) was also providing the basis necessary to support *Objectives 2* and *5* (cited below).

Objective 2: Definition of a strategic plan for the industrial development of the architecture

Objective 5: Prepare a framework guiding detailed design developments

In experiment 3.3, we have demonstrated that if the subsystem requirements are formulated with an optimization problem rather than attempting to reach conceptual targets, the subsystem developments will have a higher probability of success in achieving targets at the architecture level and will quasi-systematically achieve superior architectures. This important observation validates *Hypothesis 3.4*.

5.3.6.2 Discussion on Multi-Level Optimization Methodology

In Chapter 3 (review of the state of the art), several multi-level optimization techniques were introduced. It is important to observe that the Coordinated Optimization method differs from them on several levels.

The first level is the general purpose of method. Other multi-level optimization techniques are motivated computationally by the need to limit the exchange of information between collaborating analyses while accelerating an overall optimization process. In the case of the Coordinated Optimization, the objective is not mathematical acceleration, but rather the preparation of design conclusions and the automated setup of the sizing process.

As observed in the experiments presented earlier, the Coordintated Optimization offers properties that are not shared by other optimization schemes. The first is the simplification of the architecture analysis problem. This property results from the fact that the architecture optimization is broken down into subsystem optimization problems. As observed in experiment 2, setting up the subsystem problem implies challenges of its own, but these lower-level optimizations do simplify the architecture-level optimization problem to a bounded unconstrained problem. This objective is very different from other multi-level optimization techniques which do not necessarily attempt to simplify the

architecture-level optimization process, but rather minimize the exchange of information between collaborating analyses.

The other important difference between Coordinated Optimization and other optimization methods is the purpose of its internal information. The optimal value of the architecture level optimization parameters carry a design meaning as demonstrated in experiment 3.2. This information offers significant benefits for subsystem developments. The other forms of multi-level optimization methods are focused on the convergence of the optimization problem and do not attempt to propose further information besides the identified optimal solution.

5.3.6.3 Overview of Coordinated Optimization Benefits

Based on the experiments presented earlier, we have observed that the Coordinated Optimization approach is not only a relevant approach for sizing subsystems accurately. It is also a means to optimize the subsystem concepts towards architecture level goals and to prepare future subsystem developments. The variables used to relate the architecture and subsystem optimization levels (the priority factors) were also shown as a radically effective means to formulate subsystem requirements in a flexible but precise fashion. This problem formulation decouples the subsystem problems while maintaining their cohesion with respect to architecture objectives. This decoupling is essential to performing the clustered concurrent engineering development necessary to an industrial process. For instance, the motor designer now knows how he/she is supposed to optimize the motor. She knows how much motor weight can be traded for efficiency in order make decisions with regards to her design as opposed to trying to meet specific targets which may be suboptimal or infeasible.

As a result we can see that the CoOp method clearly addresses the fundamental Objectives of this thesis. The coordinated optimization has three major benefits:

- It enables optimizer-based subsystem sizing

- It simplifies the architecture-level analysis problem
- It formulates optimization priorities for subsystem developments

By enabling the integration of optimizer-based subsystem sizing, the coordinated optimization integrates the resolution of subsystem-level trade-offs. This resolution enables accurate sizing estimations taking into account the optimization potential of subsystems.

By relying on the subsystem optimizer (performing their sizing), the architecture-level optimization problem is significantly simplified. The subsystem-level optimizer filters out suboptimizal subsystem designs. This elimination simplifies the architecture-level optimization problem which now focuses on the identification of design priorities rather than subsystem detailed design parameters.

The identification of design priorities provided by Coordinated Optimization supplies the information necessary to steer subsystem developments toward architecture goals (i.e. defines subsystem-level criteria for contributing to architecture-level optimization). This ability both improves the performance toward architecture goals of subsystems developed with priority factors and increases the chances that subsystem development will deliver a final architecture at spec, despite development uncertainties.

Chapter 6

Proof of Concept

6.1 Introduction

This chapter will provide an architecture case study where the methodology was applied. This case study plays both the role of a proof of concept and an experiment validating the hypotheses formulated in Chapter 4.

This chapter will be initiated by a statement of objectives with regards to its double purpose (experiment and proof of concept). This introduction will also provide the reader with a general presentation of the architecture concepts explored in this case study (Architectures for the Turbo-Electric Propulsion Aircraft). This introduction is followed by a description of the preparation of design activities. This description includes a presentation of the modeling environment used in the implementation of the methodology and a description of the modeling libraries supporting it. This section is followed by a description of the baseline architecture concept. This description is based on both an informal presentation of the baseline concept and by the exposition of the SysML graphs used to define it. The next section describes the implementation of the architecture model builder (Builder). The description of the Builder is followed by the introduction of the analysis model it constructs. This section explains how the automatically produced analysis model is capable to represent the architecture concept. This chapter is concluded by a presentation of the architecture analysis activities. The methodology proposed in this thesis supports and fosters many new forms of architectural analysis. This presentation attempts to provide an overview of these new opportunities.

6.1.1 **Purpose of the Case Study**

The purpose of Experiment 4 is to test *Hypotheses 2* (including *Sub-hypotheses 2.1* and *2.2*).

Hypothesis 2: The functional, physical and operational description of the architecture provides an unambiguous description of the architecture and facilitates the establishment of the SMDA

Hypothesis 2.1: The composition of the model corresponds to the physical composition of the architecture.

Hypothesis 2.2: The functional relationships between two subsystems characterize the flow of information between their sizing processes.

An overview of this experiment is provided in Figure 170. These hypotheses have a deep role in the methodology proposed in this thesis. In order to best be able to test their validity and observe their effects on the fundamental objectives it is necessary to observe the overall process. Therefore this experiment has a double role which is to validate a set of hypotheses and to provide an overall illustration of the proposed process.

6.1.1.1 Criteria for Experiment 4

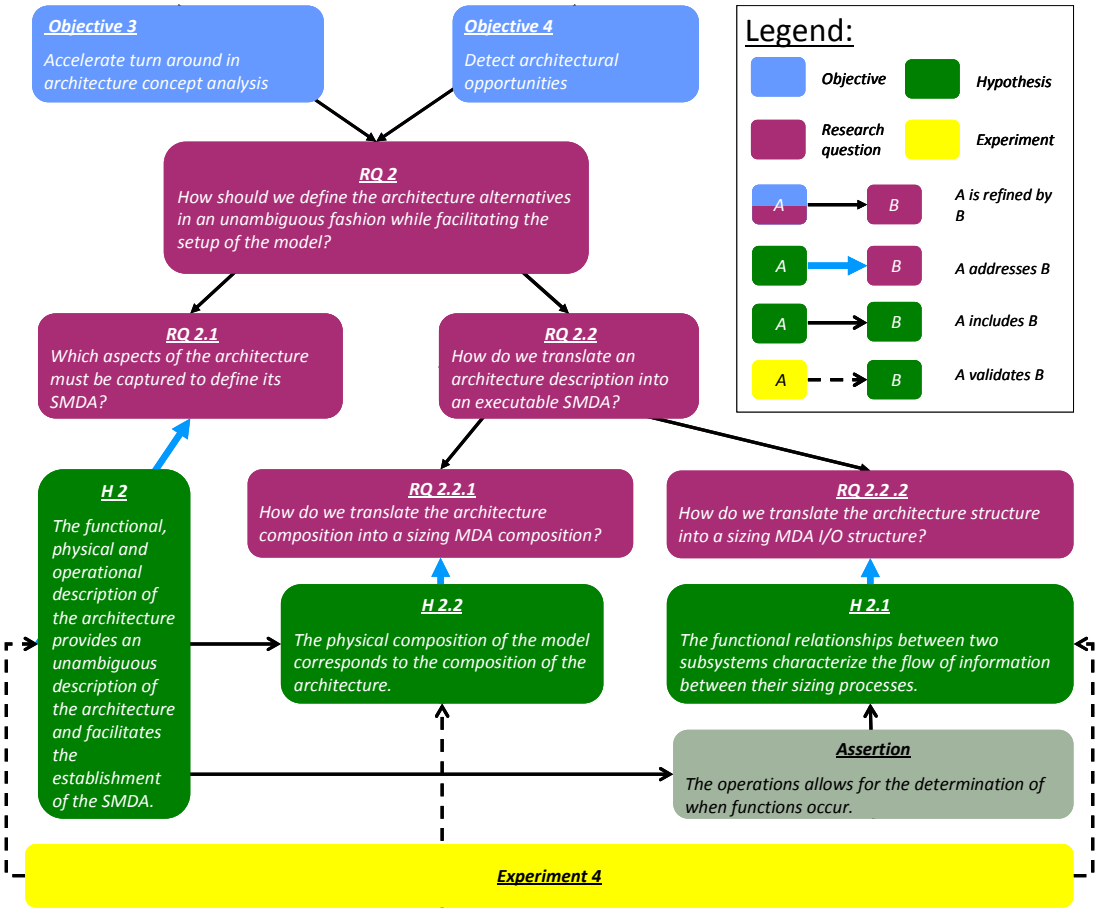


Figure 170: Overview experiment 4

In order to validate *Hypotheses 2*, *2.1* and *2.2* this experiment observes the outcome of the architecture definition process. Its observations focus on the following aspects of the architecture concept definition:

- Flexibility: Is the method capable of representing very different forms of architecture concepts?
- Clarity: Will the concept definition approach allow for an unambiguous, and visually intuitive representation of the architecture?
- Practicality: Is the number of actions necessary to characterize the architecture minimized?

- Ability to deploy in an industrial context: Can the method be easily implemented in a design office context with multiple participants accessing, defining and modifying the model?

6.1.1.2 Proof of concept for the overall methodology

The objective of the proof of concept is to present the opportunities offered by the proposed methodology. This proof will provide evidence with regards to its ability to support the fundamental objectives:

Objective 1: Provide flexible architecting methods for absorption of technological opportunities

Objective 2: Definition of a strategic plan for the industrial development of the architecture

The proof of concept is based on the case study of turbo-electric propulsive architectures. This case study will provide the reader with an example illustrating the implementation of the methodology based on a real architecture conceptual design problem. It will show how the automated creation of the analysis model can support the optimization of the architecture concept. The proof of concept will also highlight the usefulness of the conclusion of the analysis in supporting subsystem design problems.

6.1.2 **Introduction of the Case Study**

The proof of concept will present the analysis of "Distributed Turbo-Electric Propulsion" architectures. In this context, the term "distributed propulsion" specifies that the vehicle is propelled with multiple "propulsors". The propulsors are air-breathing turbo-electric thrusting devices. The term "turbo" refers to the fact that the thrust is obtained by the acceleration of a flow of air via its compression by one or multiple ducted fans. In "turbo-electric" the term "electric" refers to the means by which the fan(s) are powered. In the architecture considered, the fan(s) are driven by electric motors. These

motors are High Temperature Superconducting motors [77]. The following figures illustrate the concept with some possible aircraft configurations based on this propulsion architecture.

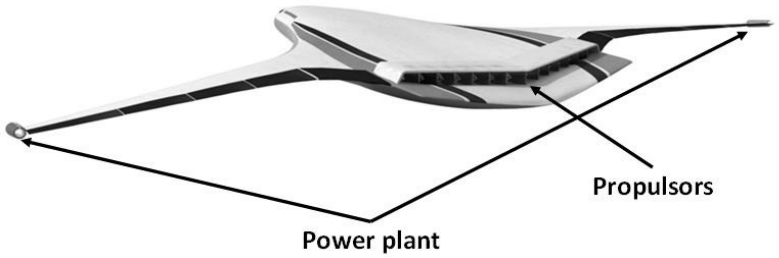


Figure 171: N2A Turbo-Electric modification presented by Felder [80]

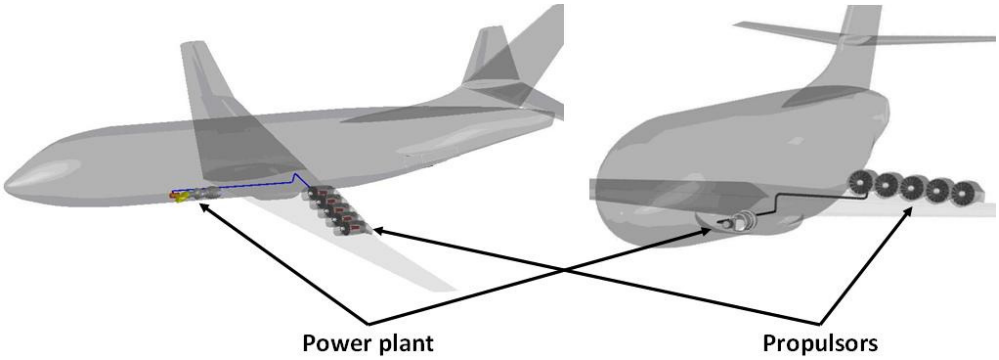


Figure 172: Turbo-Electric aircraft concept for regional jets by Mark Waters [81]

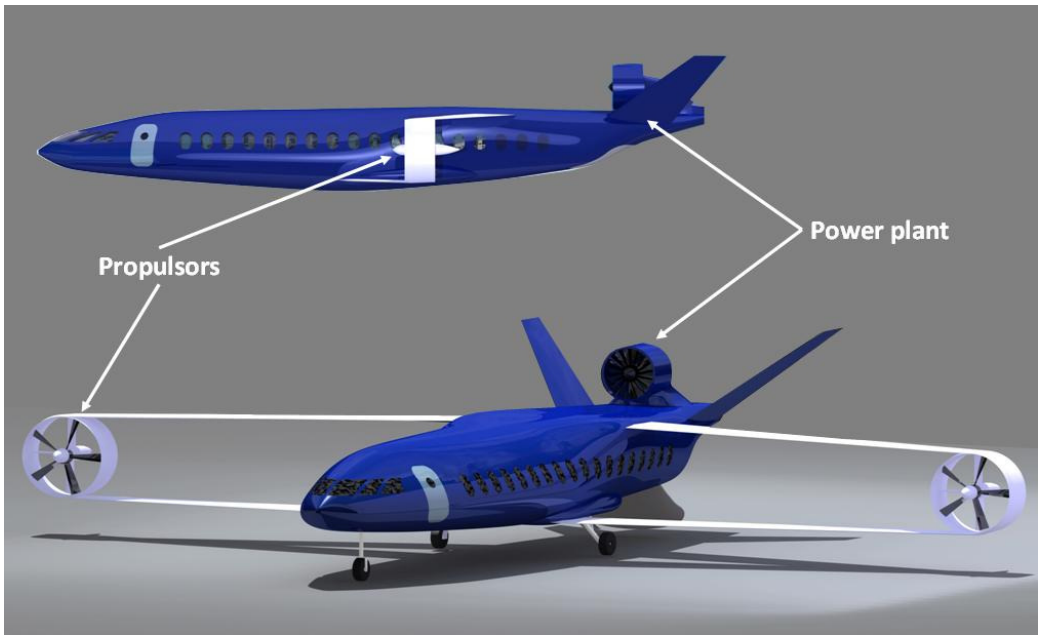


Figure 173: The "next DC-3" concept by ASDL[82]

At the time when this dissertation was prepared the distributed turbo-electric propulsion aircraft concepts were identified by NASA programs as a promising architectural/technical orientation for the future of commercial aviation [83]. The advantages include noise reduction, improved performance due to increased propulsive efficiencies and the elimination of the mechanical constraint between the fan and the turbine. But the realization of such a concept implies several important challenges both at the technological (subsystem-level) and the architectural levels. The objective of this proof of concept is to demonstrate that the methodology proposed in this thesis can successfully address some of these architectural challenges.

6.1.2.1 Novelty of the Architecture and its Failure Scenarios

One of the most critical challenges imposed by new architectures results from the diversity and novelty of their failure scenarios. Although clear performance advantages were demonstrated in previous publications [77-78, 84], these studies focus most (if not only) on normal operation without including any forms of redundancy for critical subsystems or sizing them for failure scenarios. In order for an aircraft to be certifiable (and the concept to be credible), it is essential to consider these aspects.

6.1.2.2 Intricate Sizing Process

Since this is a new architecture with new technologies and new failure scenarios the sizing rules are no longer known. In reference [85], Cumpsty affirm that for commercial transport aircrafts the propulsion system is typically sized for the top of climb thrust requirement. This rule, defined for a large bypass turbo fan, does not necessary hold in the context of this architecture. It is impossible to know, a priori, if the fan is going to sized based on climb or take-off requirements for instance. In the same fashion it is not possible to "guess" whether the power plant will be constrained in its size based on the take-off or climb requirements. This is a typical challenge common to any development of new architecture concept. Even less exotic architectures (e.g. the more

electric architectures) will impose similar challenges. The challenge lies on how to detect and formulate the set of critical scenarios which will be constraining the sizing of the subsystems composing the architecture.

6.1.2.3 Wide Architectural Design Space

Unlike traditional propulsion, electric propulsion requires a series of internal functionalities (power generation, transformation, distribution, propulsion, etc…). Each internal function can be performed in multiple ways. The redundancies can be implemented in an even greater number of possibilities. As a result, the potential for optimization of such an architecture is considerable. This challenge provides a great opportunity to demonstrate the capability to explore architectural concepts provided by the methodology.

6.1.2.4 Large numbers of Subsystem Trade-offs.

Some of the technologies integrated in the electric propulsion architecture are new to commercial aircraft applications. Some of them are known technologies which were applied to very different types of systems. For instance, the gas turbine concept used for the generation of electric power has been in existence for over 30 years. But its airborne application has been limited to Auxiliary Power Units (small engines located in the tail cone for technical and customer loads) which are optimized in a very different fashion than would be expected from the power plant for propulsive power. Similarly, the gas-turbine of a turbo-jet or turbofan would also be optimized in a very different fashion compared to the power-plant in turbo-electric propulsion architectures. Therefore, in order to make sure that we take the best advantage of the technology available, it is necessary to understand how the subsystems should be optimized. This particularity of the electric propulsion architectures makes them an interesting test case for the capabilities offered by coordinated optimization techniques.

6.1.2.5 Broad Scope of the Power Architecture

In the context of the power architecture (especially with the electric propulsion concept), the analysis of the architecture involves all the aspects of the aircraft mission. These aspects will include things going all the way from providing energy to the ovens to providing the thrust necessary for a steep climb take-off. All energy loads concern the power architecture and will influence the size of the subsystems composing it. This aspect of power architectures provides an interesting way to observe the ability of the methodology to address large and complex missions.

6.1.2.6 Description of the Design Problem Considered in the Test Case

In this test case, the investigation focuses on the optimization and development of the turbo-electric architecture concept for a single-aisle class passenger aircraft. The aircraft concept used in this test case is derived from an A320 in terms of size and base weight. In order to scope the study, the take-off growth weight is assumed to be fixed and the weight of fuel onboard is assumed to buffer the possible variations in weight.

The design problem will attempt to maximize the operational range of the aircraft. This objective implies that optimizing both the weight of the architecture as well as its efficiency. The constraints used in this mission are the certification and cut-off performance requirements discussed in Appendix A. Beyond the analysis of the performance of different architecture alternatives, this study has two overall objectives. The first is to formulate an appropriate turbo-electric aircraft architecture concept capable of performing the mission while optimizing its objectives. The second objective of this study is the formalization of subsystem requirements necessary to their future developments.

6.1.3 **Limitations and General Observations on the Case Study**

The scope of this architectural study will be limited to distributed turbo-electric propulsion concepts. This limitation in scope was a choice of necessity given the resources available for the elaboration of this proof of concept. The methodology proposed in this thesis could have captured in the same study very diverse forms of architecture including very different types of subsystems. But as it will be developed later in this chapter, each type of subsystem considered in the architectural trade must be modeled first. The development of subsystem models was performed by the author personally. Given the lack of technical and human resources (subsystem expertise) necessary to create the subsystem sizing models the author preferred to focus his efforts on the architecting methodology rather than on the development of dummy subsystem sizing models.

Due to the specific challenges described in the previous subsections, Turbo-Electric propulsion architectures make up a novel and fascinating problem which highlights the challenges that would occur in a similar fashion in the conceptual design of more traditional architectures.

6.1.4 **Design Problem Formulation for the Proof of Concept**

In this proof of concept the overall objective is to find the architecture which will maximize the aircraft operational range while being able to perform all functional requirements implied by the mission of a commercial passenger transport mission. This overall design problem implies the optimization problem indicated by equation (52). This optimization process attempts to identify the architecture concept (X_{arch}) and the set of priority factors $(\Gamma_{\bullet})$ which will optimize the subsystem sizing toward the maximization of range at the architecture level. With the framework proposed in this thesis, this optimization is performed in two parts. The first part concerns the definition of the architecture concept (X_{arch}). This definition process is performed manually by the

architecting team. This definition process is defined as an optimization process because it attempts to identify the "best" architecture concept. The other part is automatically formulated and solved. It concerns the optimization of the priority factors $(\Gamma_\bullet)$ toward the aircraft level objective of maximum range. It is important to understand that the model-based architecting process allows for the resolution of this optimization problem. It does so by setting up automatically the equations constituting the optimization problem (54) based on the formulation of the architecture concept (X_{arch}).

$$\underset{\Gamma_\bullet, X_{arch}}{Max} \quad OpRange\left(W_\bullet, \overline{FB}_\bullet\right) \tag{52}$$

Subject to:

- $\left[\overline{R}_\bullet, \overline{Spe}\right] = Structure\left(\overline{R}_{mission}, \overline{Spe}_{mission}, X_{arch}, \overline{IndFctReq}_\bullet\right)$
- $\left[W_n, \overline{FB}_n, \overline{IndFctReq}_n\right] = SubsysSizing\left(\Gamma_n, \overline{R}_n, \overline{Spe}_n\right)$ for $n \in [1, N]$

Where

OpRange	Operational range of the aircraft
X_{arch}	Description of the architecture concept (This variable which is not a numerical parameter is controlled by the architecting team).
Γ	Priority factors
W	Subsystem weight
$\overline{FB}$	Fuel burn
$\overline{R}$	Functional requirements
$\overline{Spe}$	Functional specification and operating conditions
$\overline{IndFctReq}$	Induced functional requirements
N	Total number of subsystems
Structure()	Function specified by the architecture structure formulating the requirements for each subsystem.

Symbols:

$\bar{\#}$ The type of variable # is specified for each operating state in the mission

$\#_{\bullet}$ This variable refers to all parameters of type # (e.g. $W_{\bullet}$ refers to the weight parameters of all subsystems)

$\#_n$ This variable refers to parameter # for subsystem n

6.2 Preparation of the Design activities

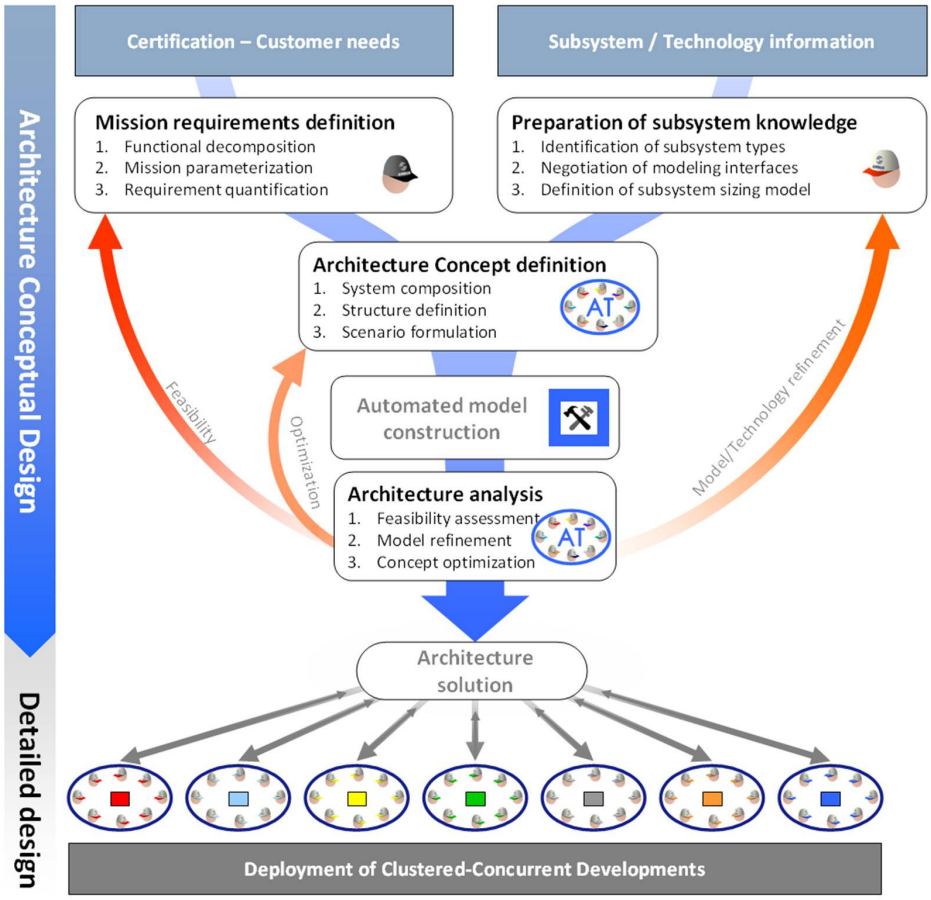


Figure 174: Overall process implied by the methodology

This section is dedicated to the preparation phase of the design activities associated with the test case. This description will be initiated by the introduction of the modeling and data framework. This introduction will present the modeling environments used in the implementation of the methodology. Based on this introduction of the modeling framework, this section will proceed with the implementation of the "Mission Requirement Definition" and "Preparation of Subsystem Knowledge" tasks.

Note: The architecture model builder, which chronologically is developed as part of the preparation activities, will be presented later in this chapter (once the reader is introduced to the basic principles used in this implementation).

6.2.1 **Introduction of the Modeling Framework**

Before introducing the tasks performed by the architecture team to prepare the architecting trades, it is necessary to understand how the overall process is implemented in terms of modeling environments and management of the information. This subsection, along with the next, provides the background necessary to understand the implementation of the methodology.

In the methodology proposed in this thesis, the term "model" (or by extension "modeling") is used to describe two different things. The first use of the term "model" is the conceptualization of subsystems and architectures. This type of model is implemented in SysML and will describe architecture concepts (e.g. the architecture includes 2 engines and 4 batteries, etc…). This type of model will be referred to as a **conceptual model**.

The other type of "model" is used for the numerical analysis of the subsystems or the architecture. This type of model sizes and assesses the performance of the physical elements composing the architecture (e.g. If the engine must provide 30,000 lb of thrust, its diameter must be 6 feet and fuel consumption 20,000 lb per hour). This form of models will be referred to as **analysis models**.

The description of the modeling framework is broken down into three parts. The first part will describe the environment used in the implementation of the conceptual model in SysML using MagicDraw. The second part will focus on the implementation of the analysis models using the Phoenix Integration suite (Model Center and the Analysis Server).

6.2.1.1 Environment for the SySML Definition of Architecture Concepts

The SysML implementation is performed in a commercially available SysML environment called MagicDraw. MagicDraw allows for the implementation of SysML diagrams. It does so by facilitating the creation of the diagrams, by checking that the rules implied by the language are respected, and by saving the information described by

the user into class and object structures. This capability allows the user to define the diagrams recommended by the methodology in a visual and intuitive fashion.

Beyond its visual SysML capability, the MagicDraw environment provides two important features necessary to the implementation of the methodology. The first important feature is its API (Application Programming Interface). The API is the feature allowing the architecture model builder to access the information formulated in the SysML graphs. The second important feature is the MagicDraw Teamwork server. This feature allows for simultaneous definition of SysML diagrams by multiple users. Given the fact that the methods proposed in this thesis are expected to be implemented in an IPT context, it is necessary to be able to build different aspects of the model simultaneously both for practical (version control and accessibility of the model) and for team efficiency reasons (opportunity to define different aspect of the concept in parallel).

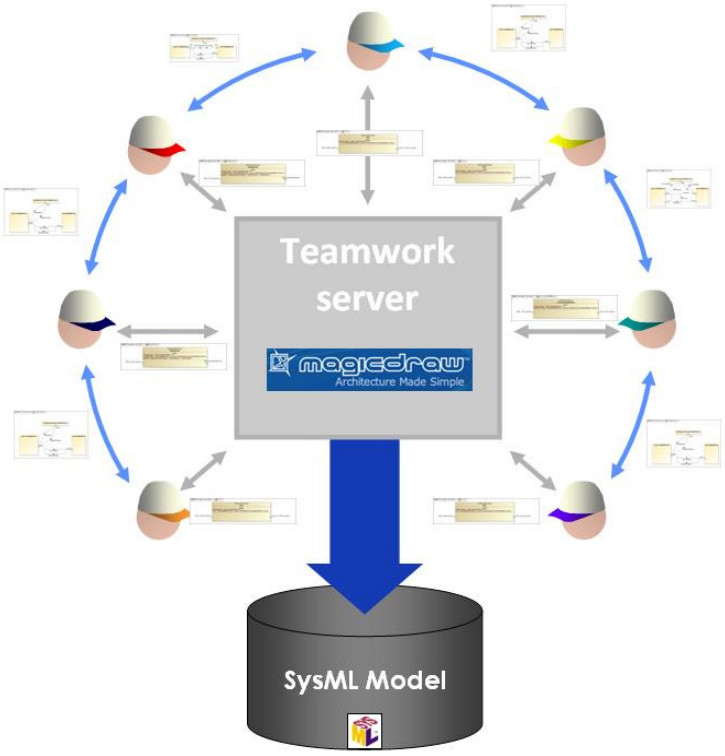


Figure 175: Role of MagicDraw

6.2.1.2 Environment for the Analysis Model Construction and Execution

The analysis models were implemented with Phoenix Integration products. This implementation was done under two aspects. The storage and management of subsystem models was performed in the Phoenix Integration Analysis Server (AS). The creation and execution of models were performed in the Model Center environment.

As shall be further described in this section, the subsystem sizing models were all coded in Matlab (for practical purposes). In order to use these models in the object-oriented fashion prescribed by the methodology, it was necessary to use an integration environment (Model Center). In order to access the Matlab functions, Model Center accesses a library via the Analysis Server. The Analysis Server library includes model components. The **model components** are analysis models with inputs and outputs. In this work, the model components correspond to the modeling bricks representing subsystem sizing models.

Since the sizing models are originally implemented in a Matlab function, the model components will correspond to objects including a reference to the Matlab function. The execution of the model component will launch a Matlab session which executes the function with the inputs specified. This process is presented in the upper part of Figure 176. This convoluted process is specified once (in the ScriptWrapper file), then this operation is automatically carried out by the Analysis Server. Therefore, in order to simplify the conceptualization of the model an abstraction is used for this modeling process. This abstraction represents the whole process as a model component provided by the Analysis server. This conceptualization is presented in the lower part of Figure 176.

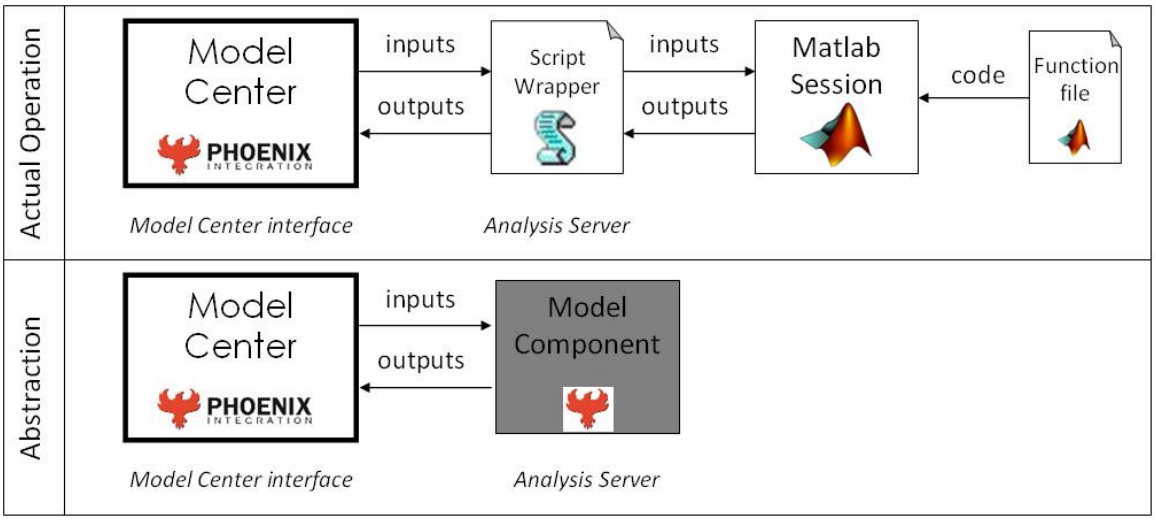


Figure 176: Conceptualization of model components

Model Center uses an object oriented approach which allows for multiple instances of the same model components in an independent fashion. Therefore, the models defined in the Analysis Server are in fact classes of models which can be instantiated as necessary.

Like MagicDraw, Model Center includes an API. From this API it is possible to access the Analysis Server library and import its model components. It is also possible to define connections between the model components. Therefore, using the API it is possible to automatically construct a Model Center model by integrating model components and linking them as necessary to represent the architecture concept. The API provides the gate necessary for the Builder to integrate these two environments.

6.2.2 **Mathematical Analogy for Conceptual and Analysis Models**

This conceptual model describes the architecture concept. Therefore the conceptual model is represented by the variable X_{arch} in the overall design problem - equation(54). In this model-based architecting methodology, the "translation" of the conceptual model into the analysis refers to the establishment of the following optimization problem:

$$\underset{\Gamma_\bullet}{Max} \; OpRange\left(W_\bullet, \overline{FB}_\bullet\right) \tag{53}$$

Subject to:

- $\left[\overline{R}_\bullet, \overline{Spe}\right] = Structure\left(\overline{R}_{mission}, \overline{Spe}_{mission}, X_{arch}, \overline{IndFctReq}_\bullet\right)$
- $\left[W_n, \overline{FB}_n, \overline{IndFctReq}_n\right] = SubsysSizing\left(\Gamma_n, \overline{R}_n, \overline{Spe}_n\right)$ for $n \in [1, N]$

Note: the meaning of the parameters used in this equation is provided in page 316.

This optimization problem is, in fact, a mathematical representation of the integrated analysis model. The model includes the subsystem sizing models (*SubsysSizing*) which are integrated by the *structure*() functions which define the mutual constraints between subsystems. It is important to note that the analysis model allows for the numerical resolution of the optimization for a given architecture (fixed X_{arch}).

In order to guide the reader through the model-based architecting process, the description of each activity will refer to this mathematical formulation in order to describe which element is being described. In this section which describes the preparation of the architecture design activities, will explain how the generic parameters are of equation (55) are defined.

6.2.3 **Mission Requirements Definition**

Figure 177: Process Overview – Mission requirement definition

The mission requirement definition was presented in the previous chapter, as part of experiment 1. The quantified requirements are presented in Appendix A. The requirements associated with the mission were implemented in a Matlab function. This function was integrated in Model Center as an analysis component (using the Matlab plug-in). This analysis component will be referred to as the "Mission block". The inputs to the mission block are the information qualifying the scenarios:

- Scenario: The name of the scenario
- Criticality: The criticality classification of the scenario
- FlightPhase: The flight phase at which the scenario occurs

In addition to the mission requirements, the mission function will also provide some miscellaneous information necessary to most subsystems sizing. This information includes:

- The duration of each operational scenario (DT)
- The rank of the operational state corresponding to a normal cruise with no failure (Cruisek)

The mission model is based on a set of conditional statements. These conditions retrieve the correct functional profile corresponding to the description of the scenario. The functional profiles are defined under the form of indexed tables defined in Appendix A. Based on this approach, represented in Figure 178, the *Mission* block was implemented using a Matlab script.

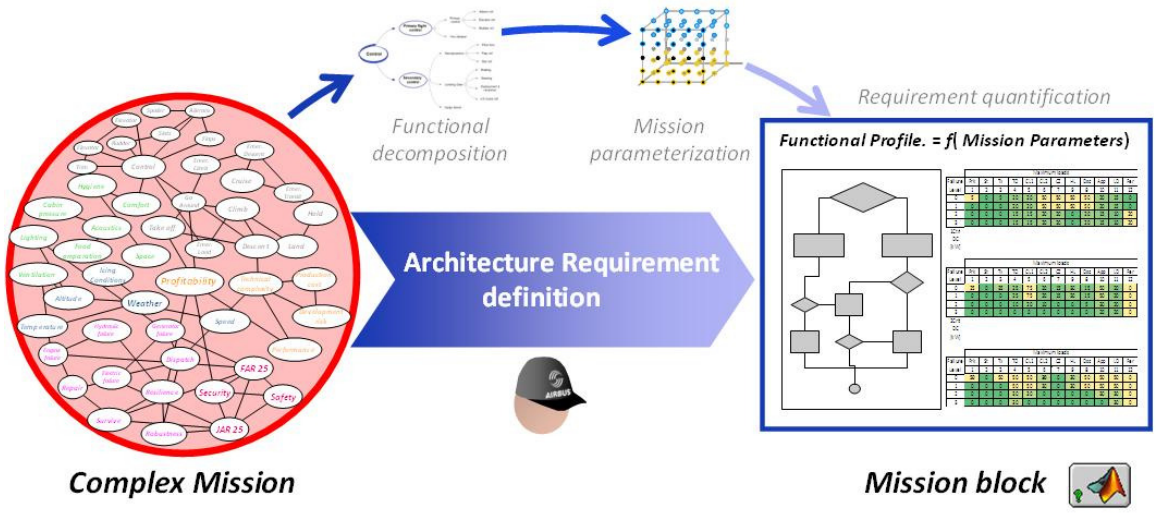


Figure 178: Implementation of mission requirement definition

The mission block allows formulating all the information related to the parameters $R_{mission}$ and $Spe_{mission}$ in equation (54).

6.2.4 **Preparation of Subsystem Knowledge**

The objective of the subsystem knowledge preparation is to prepare the SysML and sizing models necessary to the representation of each subsystem type. This preparation will be presented around the three tasks structuring the "Preparation of Subsystem Knowledge" activity and highlighted in the process figure below.

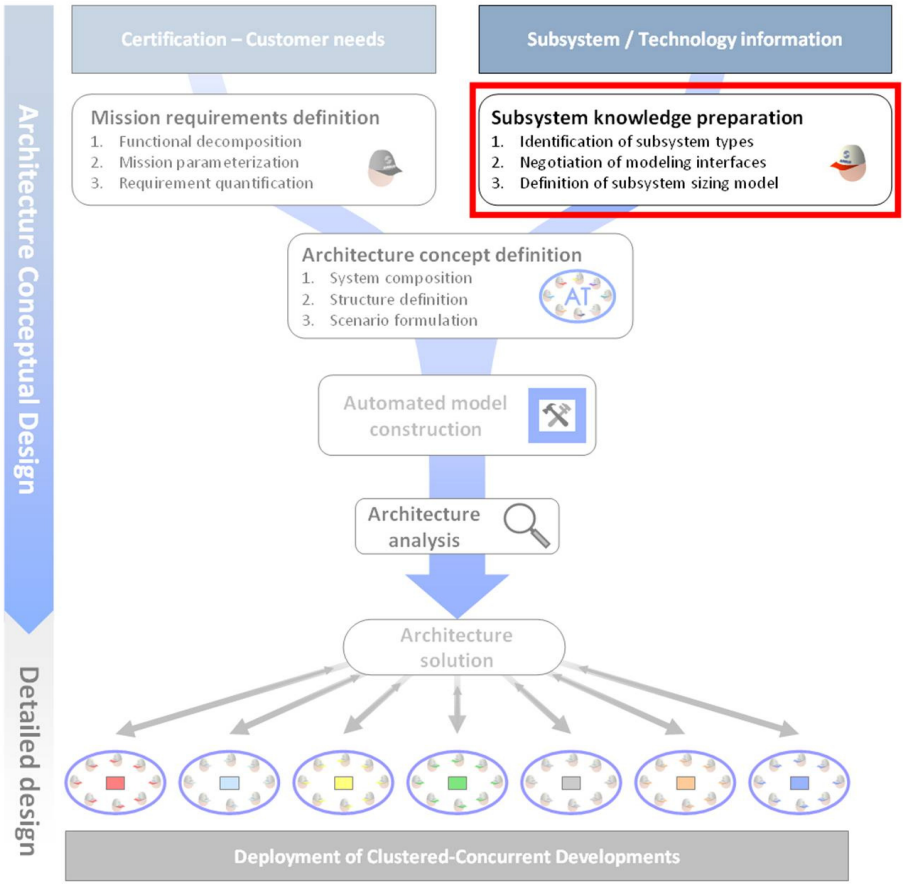


Figure 179: Process Overview - Subsystem knowledge preparation

6.2.4.1 Identification of Subsystem Types

The declaration of the subsystem types is implemented in SysML using bloc definition diagrams (as indicated in the methodology chapter). In this section, we describe the techniques used to implement the methodology. As part of this task the subsystem experts provide five pieces of information:

- The name of the type
- The functional interfaces (functional analysis of the subsystem type
- The list of non-functional attributes returned by the subsystem model
- The size of priority factors required by the sizing model (if the sizing model is of the optimizer-based type)
- The name and reference of the sizing model in the analysis server library

The SysML declaration technique associated with these five aspects is presented in the following paragraphs. A step by step description of this implementation process in MagicDraw is provided in Appendix J (userguide for the implementation of new subsystem types).

6.2.4.1.1 Declaration of Subsystem Type Name

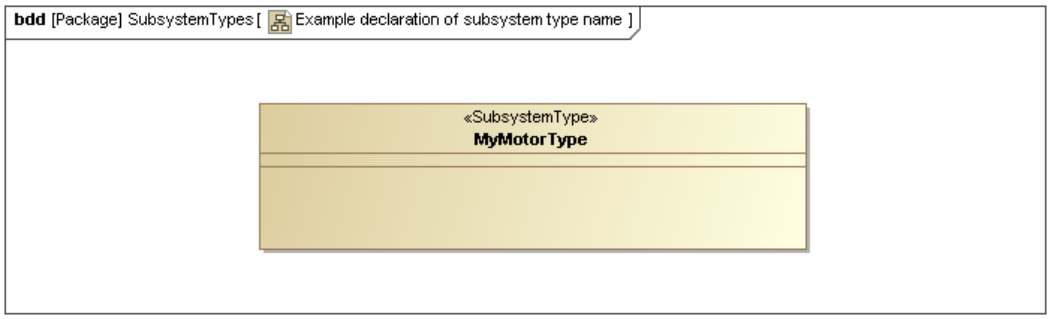


Figure 180: Example declaration of subsystem type

The subsystem type is represented by a SysML block[3]. The name of the block corresponds to the name of the subsystem type. This block becomes its representation. Every time a subsystem of this type is integrated in an architecture, a reference to this

[3] In Figure 180, the <<SubsystemType>> stereotype was applied. This application is optional. If it is not applied, the block representing the subsystem type would then read <<block>> instead of <<SubsystemType>>

block will be necessary. Similarly every time we want to add, remove or change the modeling information qualifying this type, an action must be performed on this block. The following paragraphs will describe the format used to describe this information.

6.2.4.1.2 Functional Interfaces

The subsystem type is capable of providing a function for the architecture. This capability is materialized by an outbound flow port (which will be referred to as the **function port**). If this subsystem also induces functional requirements, they will be represented by an inbound flow port. In order to work successfully with the Builder the capability port must carry a name starting by the string "*Cap*". Similarly the requirement port name must start by "*Req*". If a subsystem has multiple functional capabilities or requirements these ports should be differentiated using different suffixes.

In addition to naming the functional port, a functional flow has to be assigned. The functional flow are designated as SysML blocks. The definition of these blocks is expected to occur in parallel (or iteratively with the subsystem declaration). A description of the definition of functional flows is provided later in the "Negotiation of Modeling Interfaces" subsection (page 336).

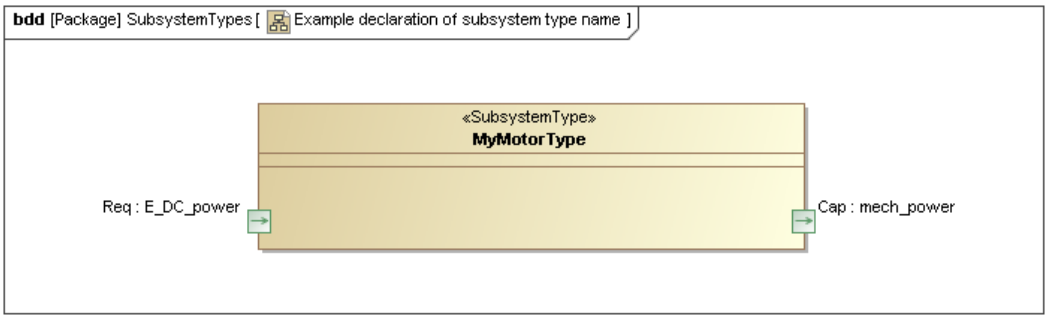


Figure 181: Example declaration of functional ports

In the example provided above the subsystem type is assumed to correspond to an electric motor. A motor is expected to provide mechanical power. Therefore, a function port is created to represent the capability of the subsystem type to provide a function. The functional port (represented on the right boundary of the block) is oriented outbound and

its name starts by "*Cap*" and is assigned the type "*mech_power*". The text next to the port indicate the name and type of the port using the following convention: [*name*]:[*name of the type*].In order to provide mechanical power, the motor will induce the need for electric power (DC in this example). Therefore, a second function port oriented outbound and typed *E_DC_power* (name of the functional flow corresponding to DC electric power) is needed. The name of this port is "*Req*" and is represented on the left boundary of the block.

6.2.4.1.3 Subsystem Attributes

Besides providing the information necessary to specify its functional requirements, the subsystem sizing model will define attributes of the subsystem which are necessary to the synthesis of the system. For instance, weight is a piece of information that must systematically be made available to the synthesis analysis. In order to identify this type of outputs, a value property is implemented. The value property representing an attribute is defined by the following three properties:

- Type: The type will inform the Builder that the value property specifies an attribute. In this test case two types were defined. The first type is "*SubsystemAttribute*". This type designate attributes of form double (i.e. attributes characterized by one value). Some examples of attributes that correspond to this type are Cost, or Weight. The second type corresponds to "*SubsystemAttributeVector*". This type is used for attributes that must be defined for each scenario (e.g. drag, fuel burn, heat rejection, etc…).
- Name: It defines the name of the input array corresponding to this attribute in the system synthesis component in model center
- Default value: It defines the name of the output variable in the model representing the subsystem.

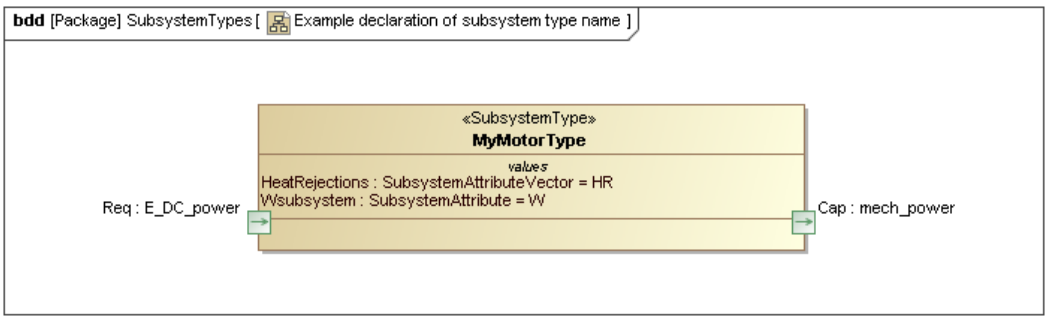


Figure 182: Example declaration of attributes

In order to illustrate the representation of the subsystem attributes in SysML, two examples were prepared. The first example is a vector attribute: "*HeatRejection*". The value property representing this attributes is listed first in the "*MyMotorType*" block representation in Figure 182. This attribute is defined for each operating state in the mission. Therefore it is represented by a vector. The name of the variable producing this vector is named "*HR*" (hence the use of the "*SubsystemAttributeVector*" type). This variable will be an output of the sizing model of the subsystems of type "*MyMotorType*". The value of this variable is destined to an array in the system synthesis analysis called "*HeatRejections*".

The second example characterizes a double attribute. This attribute is captured by the value property "*Wsubsystem*". Its type is "*SubsystemAttribute*" because the weight is characterized by a single variable. The default value is set as "*W*" because the model will be outputting a variable "*W*" indicating the weight of the sized motor. Since this attribute must be sent to the input array "*Wsubsystem*" into the system synthesis analysis, the name of the value property is indicated as "*Wsubystem*".

6.2.4.1.4 Identification of Priority Factors

If the subsystem type is sized with the optimizer-based approach, the sizing model must be connected to the architecture optimizer. In order to identify to the Builder that this connection is necessary, the subsystem type will represent this item by a value property. Similarly to the definition of the attributes, the declaration of the priority factors is defined under three properties:

- Type: The type identifies the value property as the declaration of a priority factor. The type used for this declaration is the "*OptimizationParameterType*".
- Name: For the priority variable, the name of the value property indicates the name of the input variable in the sizing model which is expected to receive the priority variables.
- Value: The value indicates the size of the priority variables (i.e. the number of terms composing the subsystem objective function)

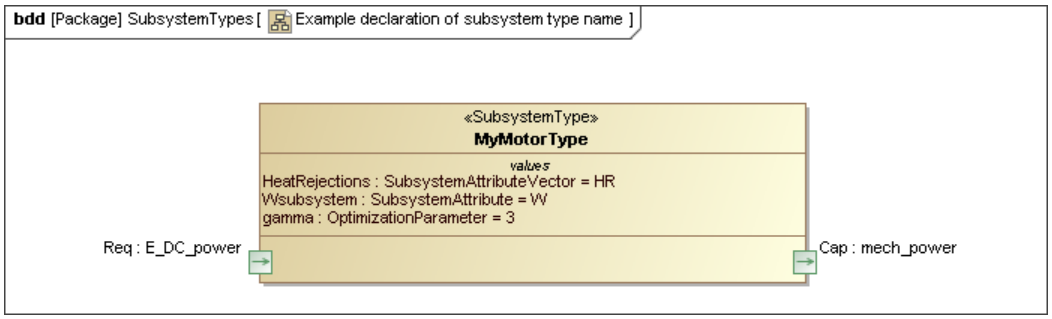


Figure 183: Example declaration of the priority factors

In the example presented in Figure 183, the last value property declares that the model used to represent *MyMotorType* will receive three priority factors. The input variables *gamma* to the model representing *MyMotorType* will receive these priority factors.

6.2.4.1.5 Designation of the Sizing Model

As we will see in the next section, "Definition of Subsystem Sizing Models" the sizing models corresponding to each subsystem type are stored in the Analysis Server Library. In order for the Builder to know which sizing model corresponds to this type, a value property is used to designate it. This value type is defined under three properties:

- Type: The type used for the sizing model is "*MCmodelReference*". This type identifies the value property as the designation of the sizing model.
- Name: The name of the value property contains the name of the sizing model in the AS library.
- Value: The value contains a string which indicates the folder path to the model inside the AS library.

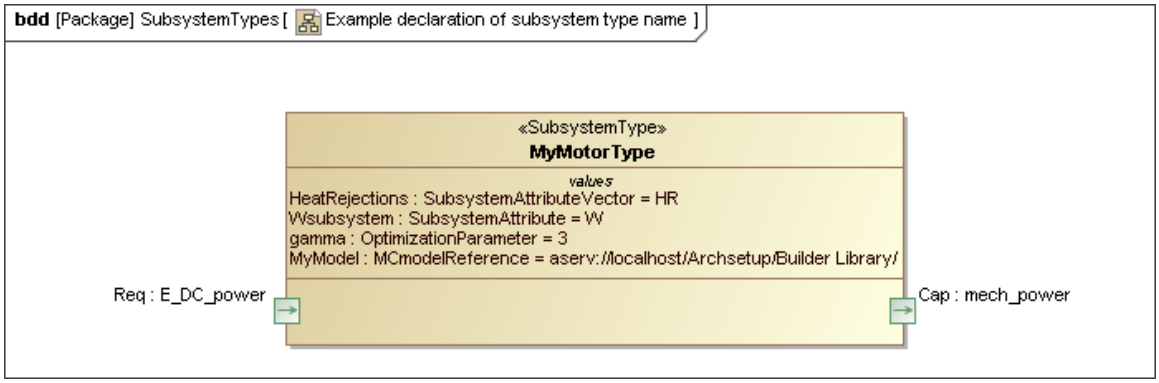


Figure 184: Example designation of the sizing model in the AS library

The example above specifies that subsystems of type "*MyMotorType*" are sized by a model named "*MyModel*". This model is located in the AS server library under the path "*localhost/Archsetup/Builder Library/*".

6.2.4.1.6 Presentation of Subsystem Types Composing the Test-Case Library

Let us now review the list of subsystem types considered for the test-case. A list including the type name and description is provided in the following table.

Table 27: Subsystem type general description

Name	Description
FFACBus *VFACBus* *DCBus*	These devices are composed of a transmission bus, and six optional transformation elements. The *FFACBus* will transmit Fixed Frequency Alternative Current (FFAC) power. The VFAC will transmit in a Variable Frequency Alternative Current (VFAC) power. The *DCBus* transmits Direct Current power. The three bus types can receive and deliver any form of electric power. The presence of the transformation elements will be triggered by a connection to a source or load using a power form different from the one used by the bus.
DDG	The Direct Current Generator (DCG) is a generator including a cyclo-convertor. This device will produce DC power and will induce mechanical power requirements.
VFG	The Variable Frequency Generator (VFG) will produce VFAC power and will induce mechanical power requirements
PowerPlant	The power plant subsystem is capable of providing mechanical power in exchange for fuel
ConventionalBattery	The battery is an energy storage device which can operate in three modes. The first mode is a “source” mode where the battery delivers power. The second mode is a “recharge” mode where the battery induces a power requirement (in order to recharge its capacity). The third mode corresponds to the “relay” mode where the battery is relaying the power received from its source to its load while replenishing its charge. In this mode, the battery has the capability to deliver power while inducing a power requirement.

ElectricFan	The electric (ducted) fan is a turbo-electric propulsor. It provides propulsive force (thrust) in exchange for electric power (assumed DC power); the performance of the electric fan is strongly dependent on the ambient operating conditions (Mach number and altitude).

Using the technique presented earlier in this subsection the types were identified in SysML. The graph used to represent these subsystem types is presented in Figure 185. In this figure, we can see that the buses have six functional interfaces (i.e. they can receive and deliver any type of electric power). This was done in order to leave the option for the architecting team to connect the bus to any of these types of power. The rule concerning the presence or absence of transformation devices correcting the type will be managed directly by the sizing model (refer to Appendix D for more information).

The battery is defined as having both the capability to deliver power and to induce the need for power. This symmetry is implemented to identify the ensemble of possible functional relationships associated with the battery. The operating mode (source, recharge or relay) will be managed by the model by detecting which connections are active for each operating state (refer to Appendix F for more information).

Among these models, two models are using the optimizer-based sizing approach (the electric fan and power plant). For simplicity sake, the others are sized based on a direct functional regression approach.

Figure 185: SysML declaration of subsystem types for test-case

6.2.4.2 Negotiation of Modeling Interfaces

In the definition of subsystem types, functional flows were used to specify their functional ports. These functional flows characterize the typical exchange of information associated with a functional relationship. This task is performed collaboratively by the architecting team composed of the experts (as illustrated in the figure below). This task is collaborative because the formulation of functional flows must be representative of the variable interfaces necessary to size the subsystems.

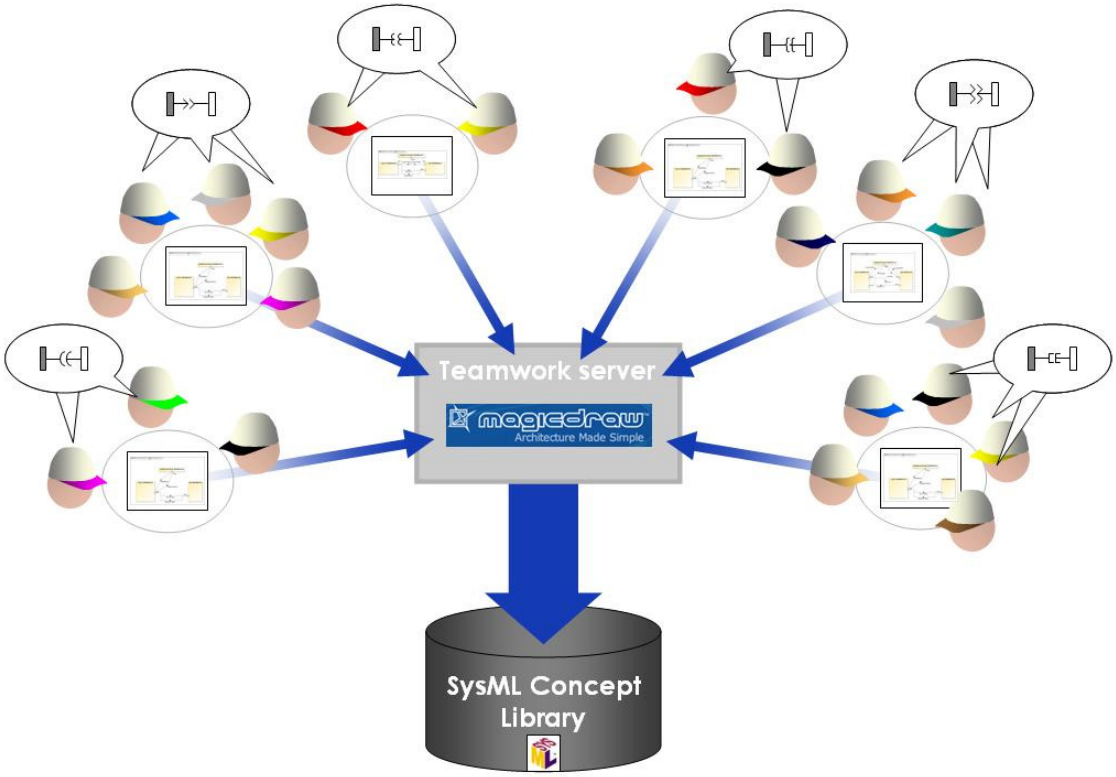


Figure 186: Process illustration – Negotiation of modeling interfaces

This subsection describes the implementation of the methodology in defining functional flows. In order to introduce this implementation, we shall first review the fundamental methods and formalisms supporting the definition of functional flows. Based on this review, we will observe that all function flows will share a common part to their definition. The definition of this common part is presented in the second paragraph of this section. The description of the implementation techniques will be concluded by

the description of how the variable naming convention is defined using internal block definition diagrams. This section will be concluded by a review of the definition of the functional flows used in the test-case.

6.2.4.2.1 Brief Review of the Methodology

All functional flows are defined on a similar base. This base was described in the methodology section in chapter 4 and is described graphically in Figure 187: General description of functional flow structure (reproduction from Chapter 4).

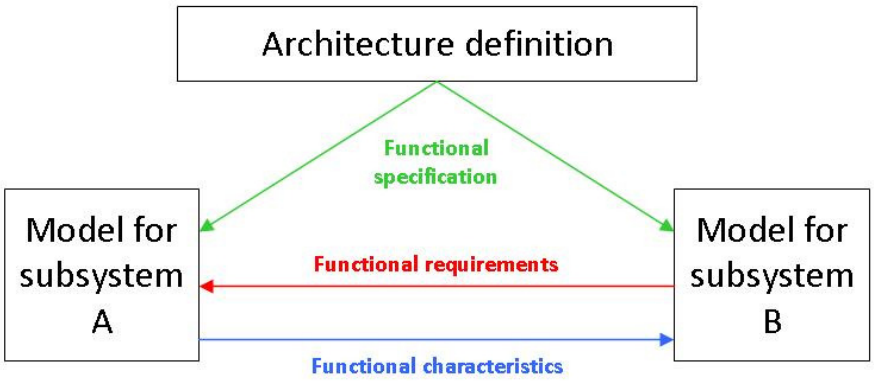


Figure 187: General description of functional flow structure

Functional flows characterize the exchange of information between three parties:

- The source (i.e. the subsystem providing the function)
- The load (i.e. the subsystem or mission requirement requesting the function)
- The architecture definition (i.e. the prescriptions made by the architecting team when defining the functional relationship)

When defining the functional flow we are interested in describing the exchange of information between the models representing each of the parties listed above. Therefore, the flow of information must be defined with regards to the following properties:

- The type of information exchanged
- The origin of the information (which party is providing this information)
- The destination of the information (which party is expecting the information)

- The name of the variables (both at the origin and the destination)

The description of the implementation technique used in this test-case is presented in the following three paragraphs. The paragraph “Definition of the Generic Functional Flow” describes the initialization step of the functional flow. The implementation of the first three points listed above (flow name, origin and destination) is described in the paragraph “Declaration of Information Flows”. The definition of the naming convention for functional flow variables concludes the description of the implementation.

6.2.4.2.2 Definition of the Generic Functional Flow

All functional flows will represent exchanges between the three parties listed above. Each party is represented by a SysML part property of the functional flow. In order to avoid repeating the initialization of the functional flow, a generic functional flow is defined: “*GenericFctFlow*”. Its definition is shown in the block definition diagram presented in Figure 188.

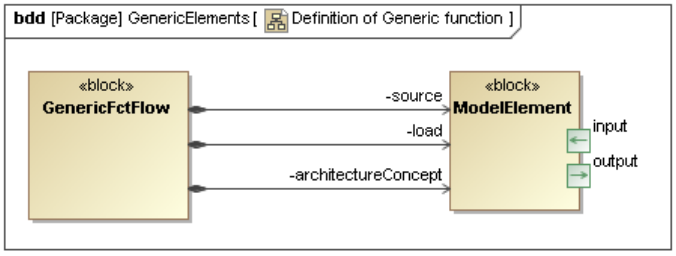


Figure 188: Definition of the generic function flow

Each of the parties is designated as a part of type “*ModelElement*”. The “*ModelElement*” type is also defined in Figure 188. It includes two flow ports. One of the ports indicates that the information connected to it will be an input to the model while the other indicates that the information will be an output. The paragraph “Declaration of Information Flows” will clarify their usage.

Based on this generic functional flow element, the functional flows defined in the previous section can be predefined. This pre-definition is performed in the diagram in Figure 189. This diagram presents the pre-definition of the five functional flows

implicitly defined previously as part of the functional interface definition of subsystem types. This graph defines implicitly that the five functional flows share the composition of the generic functional flow described above.

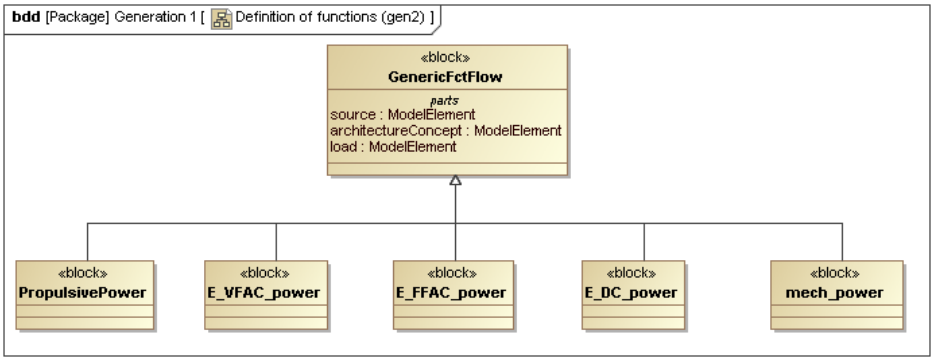


Figure 189: Pre-definition of functional flows

6.2.4.2.3 Declaration of Information Flows

In order to represent the exchanges between these parties, we use the internal block definition diagram (as defined by the methodology). This diagram is implemented for each functional flow. The connections implemented represent an exchange of information. The connections are expected to connect an "output" port to an "input" port. The model where the information originates from is the one corresponding to the party of the output port. The destination of the information is the model associated with the input port.

In the functional flow diagram all connections convey a block. The name of the block conveyed defines the type of information represented. The following diagram represents the functional flow associated with the function provide DC electric power.

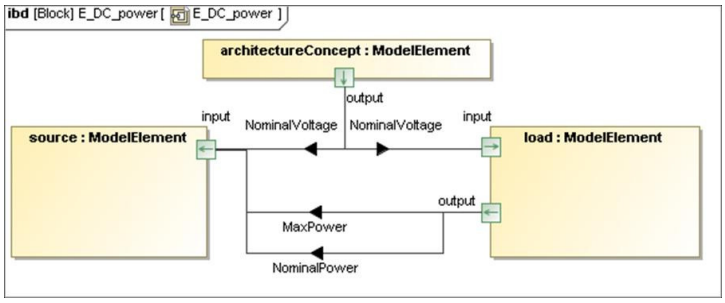


Figure 190: Example of functional flow diagram

This functional flow specifies the exchange of information associated with the "provide DC power" function. In this example four exchanges compose the flow. The first two are the "*MaxPower*" and "*NominalPower*". These names correspond to the name of the blocks conveyed by the connections going from the output of the "load" to the input of the "source". The other two connections specify that both the "source" and "load" are receiving information qualifying the "*NominalVoltage*" originating from the "*architectureConcept*".

This diagram specifies what information is being exchanged between the three parties. But in order to implement the exchange of information in the model, it is necessary to specify the name of the variable associated with each side of the relationship. In order to obtain this information, it is necessary to observe the value properties of the block conveyed by the connection.

6.2.4.2.4 Naming Convention for Functional Flow Variables

The block conveyed by the connections must include three value properties. These value properties are named "Format1", "Format2" and "Format3". Each will have a value which designates the name of the input or output associated with the exchange of information. The general rule is that:

- "Format1" defines the name of the variable associated with the model representing the source
- "Format2" defines the name of the variable associated with the model representing the load
- "Format3" defines the name of the variable associated with the model element providing the architecture concept variables.

If the load is a boundary function (i.e the functional requirements are directly defined by the mission) these rules are going to change slightly

- "Format1" still defines the name of the variable associated with the model representing the source
- "Format2" is ignored
- "Format3" will be used to define both the variable originating from the architecture concept and the variable originating from the boundary function.

In order to illustrate this rule, let us consider the variable format associated with the function "*E_DC_power*". Its naming convention is described in Figure 191.

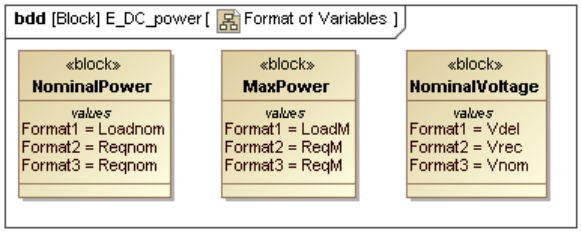


Figure 191: Example naming convention

In this example, the variables associated with a subsystem capable of providing electric DC power will be: *Loadnom* (for *NominalPower*), *LoadM* (for *MaxPower*), and *Vdel* (for *NominalVoltage*). One can easily see this by looking at all the "Format1" values. Similarly a subsystem requiring DC power will have the variable names associated with "Format2" (*Reqnom* for *NominalPower*, *ReqM* for *MaxPower*, and *Vdel* for *NominalVoltage*).

In situations were the function is associated with a boundary function, the requirements are fixed. Therefore the information flowing to the load party is ignored (in the example of the *E_DC_Power*, the variable *NominalVoltage* is ignored). The variable associated with the mission requirement will be *Reqnom* for *NominalPower* and *ReqM* for the *MaxPower*.

6.2.4.2.5 Presentation of the Functional Flows used in the Test-Case

The diagrams associated with *E_DC_power* were presented in Figure 190 and Figure 191. The diagrams associated with the other functions are presented in the following figures.

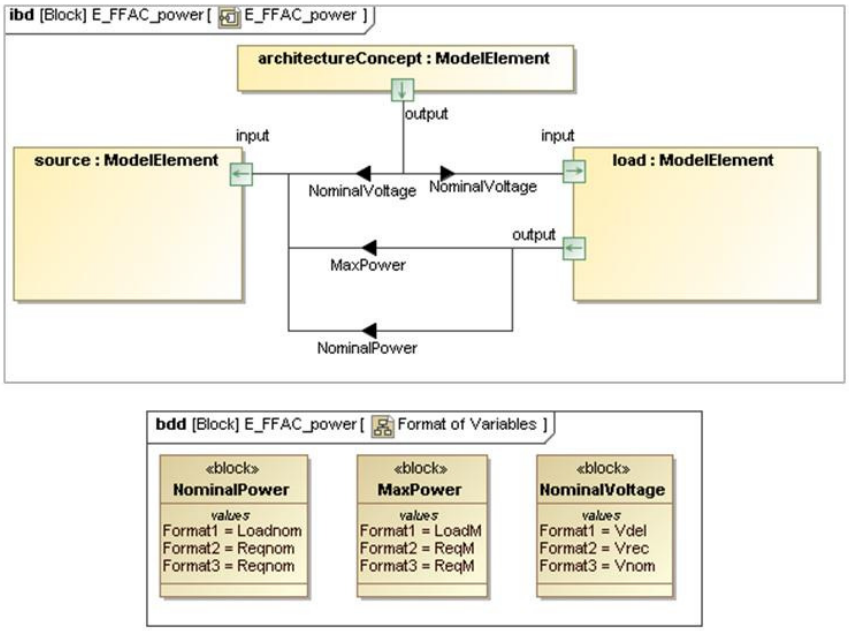


Figure 192: Definition of E_FFAC_power flow

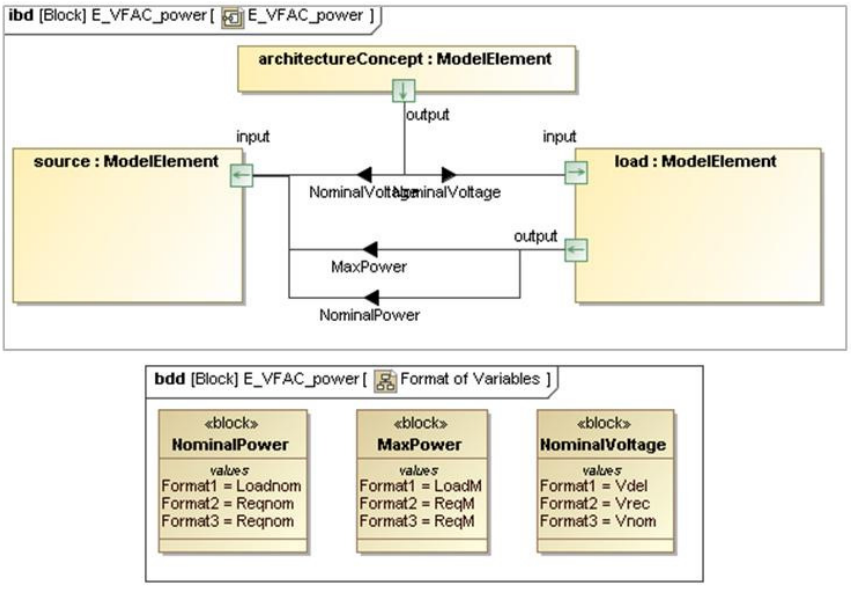


Figure 193: Definition of E_VFAC_power flow

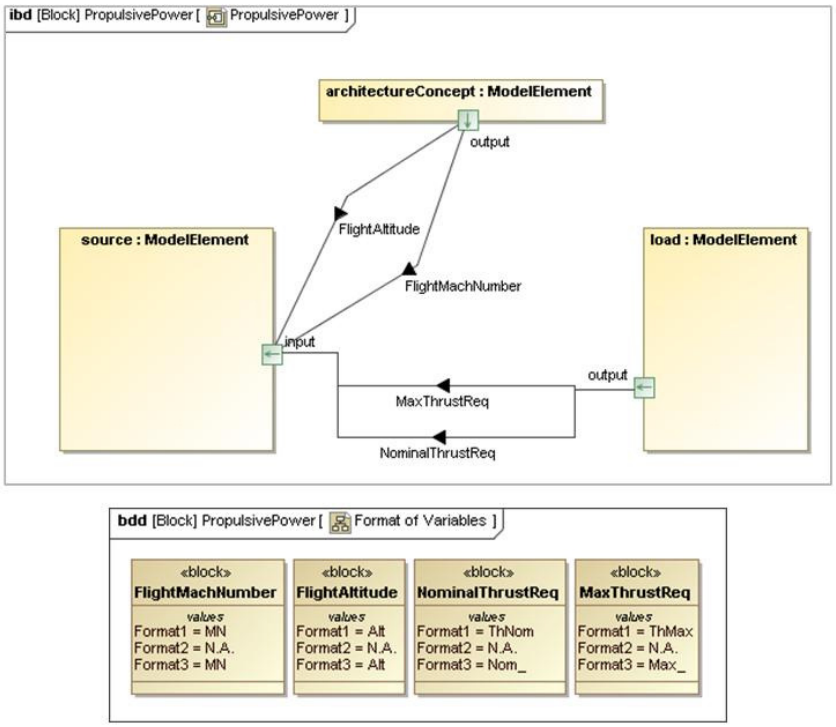


Figure 194: Definition of PropulsivePower flow

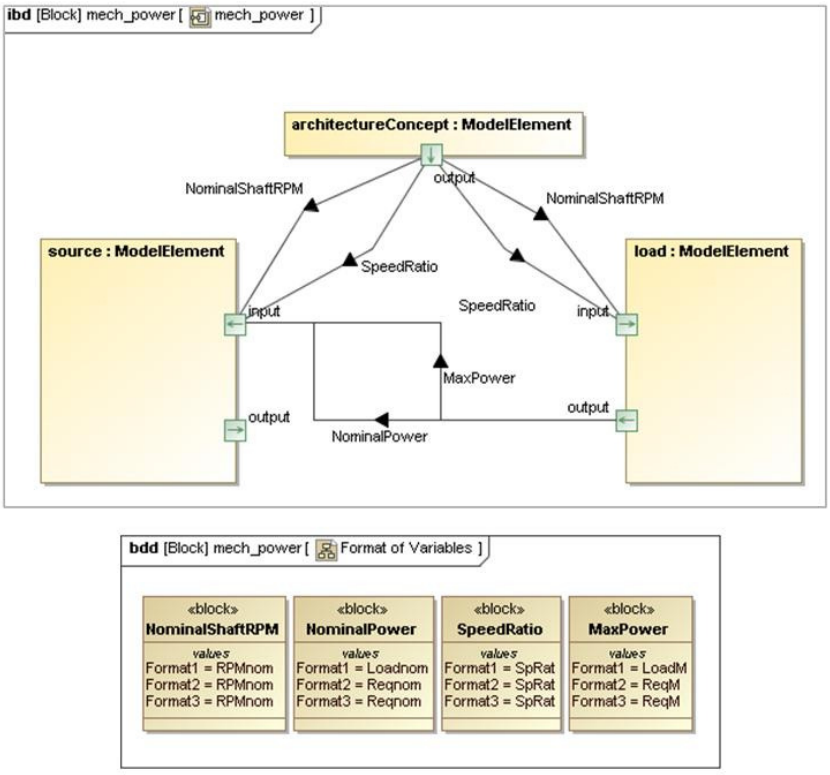


Figure 195: Definition of mech_power flow

6.2.4.3 Definition of Subsystem Sizing Models

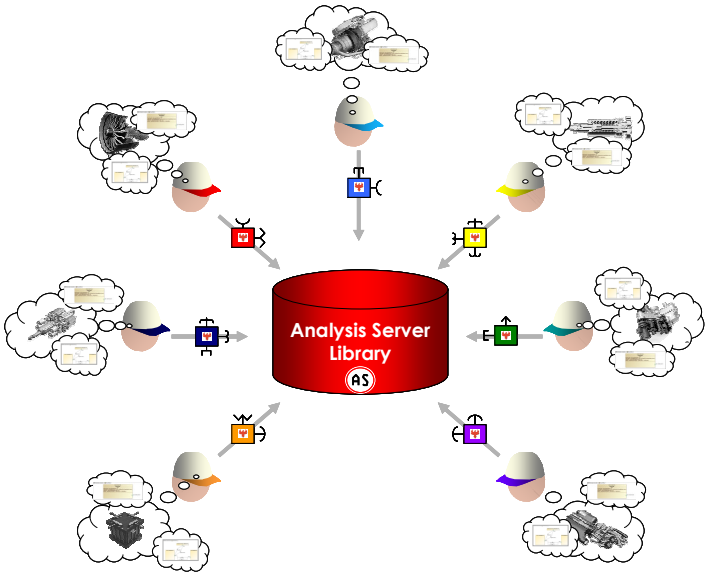


Figure 196: Process illustration – Definition of subsystem models

This section describes how the sizing models are constructed. These models are used to represent what was defined conceptually previously and will be assembled to constitute what will eventually become the architecture analysis model. The architecture analysis which is represented by the optimization problem (53) represented below.

$$\underset{\Gamma_\bullet}{Max} \quad OpRange\left(W_\bullet, \overline{FB}_\bullet\right) \tag{53}$$

Subject to:

- $\left[\overline{R}_\bullet, \overline{Spe}\right] = Structure\left(\overline{R}_{mission}, \overline{Spe}_{mission}, X_{arch}, \overline{IndFctReq}_\bullet\right)$
- $\left[W_n, \overline{FB}_n, \overline{IndFctReq}_n\right] = SubsysSizing\left(\Gamma_n, \overline{R}_n, \overline{Spe}_n\right)$ for $n \in [1, N]$

In this optimization problem the subsystem sizing model (represented by SubsysSizing) constitute the building blocks of the architecture analysis. Because of this key role, the subsystem sizing models are referred to as **modeling bricks**. Figure 196 which provides a general overview of the process highlights the key aspects of the implementation of this activity. In this activity each expert is building the sizing models. In order for these models to become compatible modeling bricks, they must be compliant

with the standards and conventions defined previously in SysML. Finally, in order to make these modeling bricks available to the Builder, they must be stored in the Analysis Server Library.

The description of this implementation process is presented in three paragraphs. The first will discuss the definition of the model interfaces (list of inputs and outputs). The second section will introduce the definition of the models and will direct the reader to the appendices describing the development of the subsystem types considered in this test-case. The third will describe the process necessary to integrate the models into the Analysis Server Library.

6.2.4.3.1 Definition of model interfaces

The two previous subsections have shown the process associated with the declaration of the subsystem and functional flow types. Based on these declarations, the inputs and outputs associated with the sizing model are predefined. The functional flows assigned to the functional ports define all variables associated with the functional interface of the subsystem with the other elements. The attribute and priority factor declarations complete the definition of the input/output interfaces of the sizing models.

The declared output interfaces constitute a “minimum” list of attributes that must be defined by the model. The declared input interfaces provide a maximum list of inputs that can be received from other subsystem. In addition to these inputs three defaulted variables were made available to the subsystem model. The first characterize the duration of each scenario (*DT*). The second identifies the rank at which the cruise scenario (design point) is located. The third element is a switch variable to display the sizing information (to let the expert know how his/her subsystem was sized). The following tables present the list of input and outputs (I/O) associated with the eight models used in the test-case. Note that the I/O list corresponding to the three buses are identical and were represented only once.

Table 28: Input / Output variables associated with the electric fan model

Variable Name	Input / Output	Type	Description
Dt	I	Generic	Time step
Cruisek	I	Generic	Rank of cruise scenario
show	I	Generic	Switch variable displaying internal information
gamma	I	priority factor	Priority factors (weight, diameter, energy, power)
Alt	I	PropulsivePower	Flight Altitude
MN	I	PropulsivePower	Flight Mach number
ThMax	I	PropulsivePower	Max thrust required
Thnom	I	PropulsivePower	Average thrust required
Vrec	O	E_DC_power	Nominal voltage of power received
W	O	Attribute	Weight of subsystem
ReqM	O	E_DC_power	Peak electric power required
Reqnom	O	E_DC_power	Average electric power required
DV_fan	O	Internal parameters (ignored)	Design variables associated with final design

Table 29: Input / Output variables associated with the gas turbine model

Variable Name	Input / Output	Type	Description
Dt	I	Generic	Time step
Cruisek	I	Generic	Rank of cruise scenario
show	I	Generic	Switch variable displaying internal information
gamma	I	priority factor	Priority factors (weight, diameter, energy, power)
T41	I	default value	Turbine inlet temperature
LoadM	I	mech_power	Peak mechanical power required
Loadnom	I	mech_power	Average mechanical power required
RPMnom	I	mech_power	Average rotational speed required from the gearbox
SpRat	I	mech_power	Maximum allowable speed ratio
W	O	Attribute	Gas turbine weight
mf	O	Attribute	average fuel flow rate

Table 30: Input / Output variables associated with the VFG model

Variable Name	Input Output	Type	Description
Dt	I	Generic	Time step
Cruisek	I	Generic	Rank of cruise scenario
show	I	Generic	Switch variable displaying internal information
VdelDC	I	E_VFAC_power	Voltage of VFAC power delivered
LoadM	I	E_VFAC_power	Peak VFAC load on generator
Loadnom	I	E_VFAC_power	Average VFAC load on generator
SpRat	I	mech_power	Maximum allowable speed ratio
RPMnom	I	mech_power	Average rotational speed required from the gearbox
SpRat	I	mech_power	Maximum allowable speed ratio
W	O	Attribute	Gas turbine weight
ReqM	O	mech_power	Peak power required from the shaft
Reqnom	O	mech_power	Average power required from the shaft
HRM	O	ignored	Maximum heat rejection per scenario
Hrnom	O	ignored	Average heat rejection per scenario

Table 31: Input / Output variables associated with the DCG model

Variable Name	Input Output	Type	Description
Dt	I	Generic	Time step
Cruisek	I	Generic	Rank of cruise scenario
show	I	Generic	Switch variable displaying internal information
VdelDC	I	E_DC_power	Voltage of DC power delivered
LoadM	I	E_DC_power	Peak DC load on generator
Loadnom	I	E_DC_power	Average DC load on generator
SpRat	I	mech_power	Maximum allowable speed ratio
RPMnom	I	mech_power	Average rotational speed required from the gearbox
SpRat	I	mech_power	Maximum allowable speed ratio
W	O	Attribute	Gas turbine weight
ReqM	O	mech_power	Peak power required from the shaft
Reqnom	O	mech_power	Average power required from the shaft
HRM	O	ignored	Maximum heat rejection per scenario
Hrnom	O	ignored	Average heat rejection per scenario

Table 32: Input / Output variables associated with the bus models

Variable Name	Input / Output	Type	Description
Dt	I	Generic	Time step
Cruisek	I	Generic	Rank of cruise scenario
show	I	Generic	Switch variable displaying internal information
length	I	default value	Length of the bus (manually defined and defaulted)
Vnom	I	default value	Nominal voltage of power transmited in the bus
VrecDC	I	E_DC_power	Voltage of DC power input
VrecFFAC	I	E_FFAC_power	Voltage of FFAC power input
VrecVFAC	I	E_VFAC_power	Voltage VFAC power input
VdelDC	I	E_DC_power	Voltage of DC power delivered
LoadMDC	I	E_DC_power	Peak DC load on bus
LoadnomDC	I	E_DC_power	Nominal DC load on bus
VdelVFAC	I	E_VFAC_power	Voltage of VFAC power delivered
LoadMVFAC	I	E_VFAC_power	Peak VFAC load on bus
LoadnomVFAC	I	E_VFAC_power	Nominal VFAC load on bus
VdelFFAC	I	E_FFAC_power	Voltage of FFAC power delivered
LoadMFFAC	I	E_FFAC_power	Peak FFAC load on bus
LoadnomFFAC	I	E_FFAC_power	Nominal FFAC load on bus
W	O	Attribute	Bus weight
ReqMDC	O	E_DC_power	Peak DC power required by the bus
ReqnomDC	O	E_DC_power	Nominal DC power required by the bus
ReqMFFAC	O	E_FFAC_power	Peak FFAC power required by the bus
ReqnomFFAC	O	E_FFAC_power	Nominal FFAC power required by the bus
ReqMVFAC	O	E_VFAC_power	Peak VFAC power required by the bus
ReqnomVFAC	O	E_VFAC_power	Nominal VFAC power required by the bus

6.2.4.3.2 Creating the Sizing Models

The description of the models is presented in the Electric Ducted Fan and Power Subsystem Sizing models appendices B through F. The process associated with the development of the electric ducted fan was discussed as part of experiment 2 presented in the previous chapter.

6.2.4.3.3 Wrapping Models for Automated Integration

All models used in this test-case were implemented in Matlab functions. In order to be able to use these models in an object oriented fashion, each Matlab model is "wrapped" and stored in the Analysis Server library. This process is automated by a script generator. This script generator was also implemented in Matlab and generates a VB script. This script creates a Model Center (MC) model object stored in the Analysis Server. The script implements this MC model by creating a reference to the original Matlab function. The name of this type of script is "*Scriptwrapper*". The code implemented for the script generator is provided in Appendix H. This appendix also includes a *scriptwrapper* example for the electric fan.

6.2.4.4 Conclusions on Subsystem Knowledge Preparation

In these pages, the presentation of the subsystem knowledge preparation is described as a linear process. In fact, this process is highly iterative. The negotiation aspects associated with this preparation activity were not discussed. The reason for using the term "negotiation" is due to the fact that different subsystems may require slightly different information from the same functional relationship. The method proposed in this thesis offers a general formulation which organizes all the effects necessary to capture sizing relationships. Therefore, as new exchanges of information are identified in functional flows new inputs or outputs must be implemented in the models. Vice versa, as models are implemented, piece of information previously overlooked may become necessary. This newly identified necessity may imply the redefinition of functional flows, which in turn will require updating the definition of the other models. This clearly represents a challenge for the implementation of the method. This problem is entrenched into the nature of complex system developments and is unavoidable (regardless of the method applied). Avoiding modeling negotiations implies neglecting technical aspects of the trade. This negligence may decrease dramatically the accuracy of the analysis and

therefore the value of its conclusions. The method proposed in this thesis provides a formal and organized context for these negotiations, hence greatly facilitating the communication process between the experts.

In order to further facilitate this negotiation process, the method would greatly benefit from a tool which automatically detects the list of input and output variables (along with their name associated with each subsystem type). In the current implementation, the information concerning the variables is scattered between the block definition diagram (used in the "Identification of Subsystem Types" task), and the internal block definition diagrams of the functional flows (defined in the "Negotiation of Modeling Interfaces" task). It would greatly simplify the process if all the information could be summarized in a single diagram or table of the type shown earlier in Table 28 to Table 32.

6.3 Definition of the Baseline Architecture

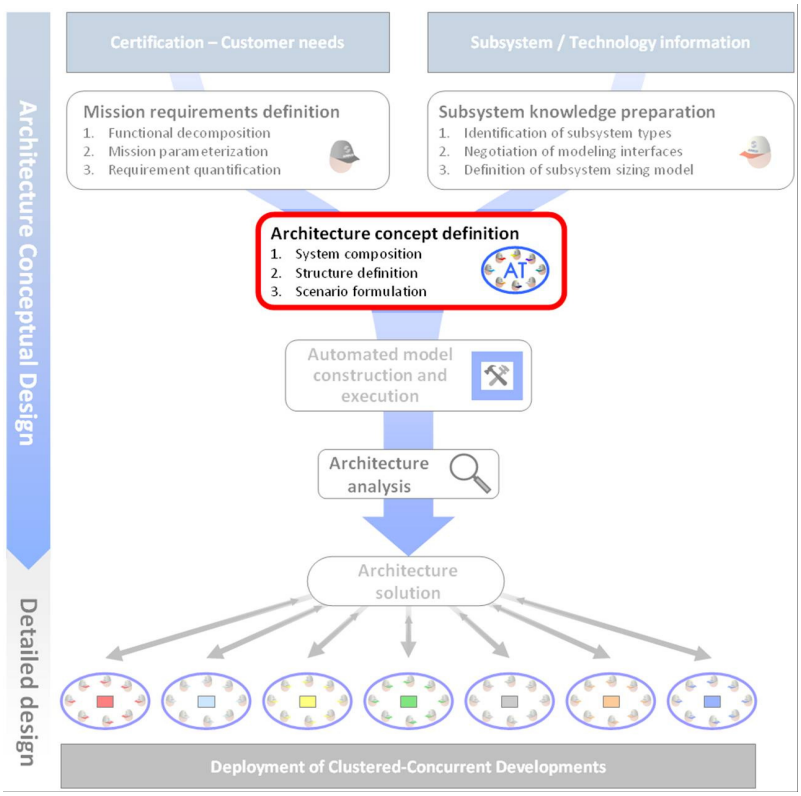


Figure 197: Process Overview - Subsystem knowledge preparation

In the previous activities, two libraries were prepared. These libraries (the Analysis Server Library and the Subsystem Model Library) provide the information necessary to construct the analysis associated with architecture concepts. This section will now present the description of an architecture concept using SysML. In order to illustrate the methodology, the turbo-electric propulsion architecture is implemented.

Before we present how the architecture is implemented in SysML, an informal presentation of the baseline architecture concept considered in the test case is provided. Based on this familiarization to the architecture concept, the reader will be able to better understand the implementation process of the architecture using the SysML language.

6.3.1 **Overview of the Architecture**

This overview presents the baseline architecture concept in an informal fashion. This information presentation will first introduce the subsystems composing the architecture along with their role in the architecture. This introduction is followed by the description of the architecture configurations in normal operation. This informal presentation is concluded by a presentation and discussion of the failure configurations considered in this proof of concept

6.3.1.1 Description of the Architecture in Normal Operation

The baseline architecture concept is composed of:

- 3 power plants: one APU and two propulsive power plants (referred to as *APU*, *PP_L* and *PP_R*)
- 5 DC generators: Each propulsive power plant is equipped with 2 DC generators (*PropGenL1*, *PropGenL2* connected to *PP_L* and *PropGenR1*, *PropGenR2* connected to *PP_R*). The fifth DC generator is an auxiliary propulsive generator powered by the *APU* (called *AuxPropGen*).
- 3 VF generators: The variable frequency generators are exclusively dedicated to non-propulsive power. There is one VF generator for each gas turbine (*APU*, *PP_L* and *PP_R*), they are respectively call *GenC*, *GenL* and *GenR*.
- 3 DC Buses: Two of these buses are energized by the main power plants. They will provide for the electric fans. They are referred as *PropBusR* and *PropBusL*. The third DC bus will support primary and secondary control power loads. This bus, referred to as the *ProtectedBus*, will receive its energy from the *APU* or the power plants depending on the flight phase.
- 2 VFAC buses: The variable frequency buses are dedicated to non-propulsive functionalities only. These buses are energized by the VF generators related to the power plants and the APU.

- 1 FFAC bus: This bus distributes the power to the non propulsive functionalities requiring FFAC power. It also supports the Protected DC bus in flight (except in take-off, landing and approach were the priority DC bus is segregated from other loads by receiving its energy from the APU).

The following figure presents an overview of the architecture configurations in different flight phases. Since the configuration changes depending on the flight phase, each configuration used in normal operation is introduced hereafter.

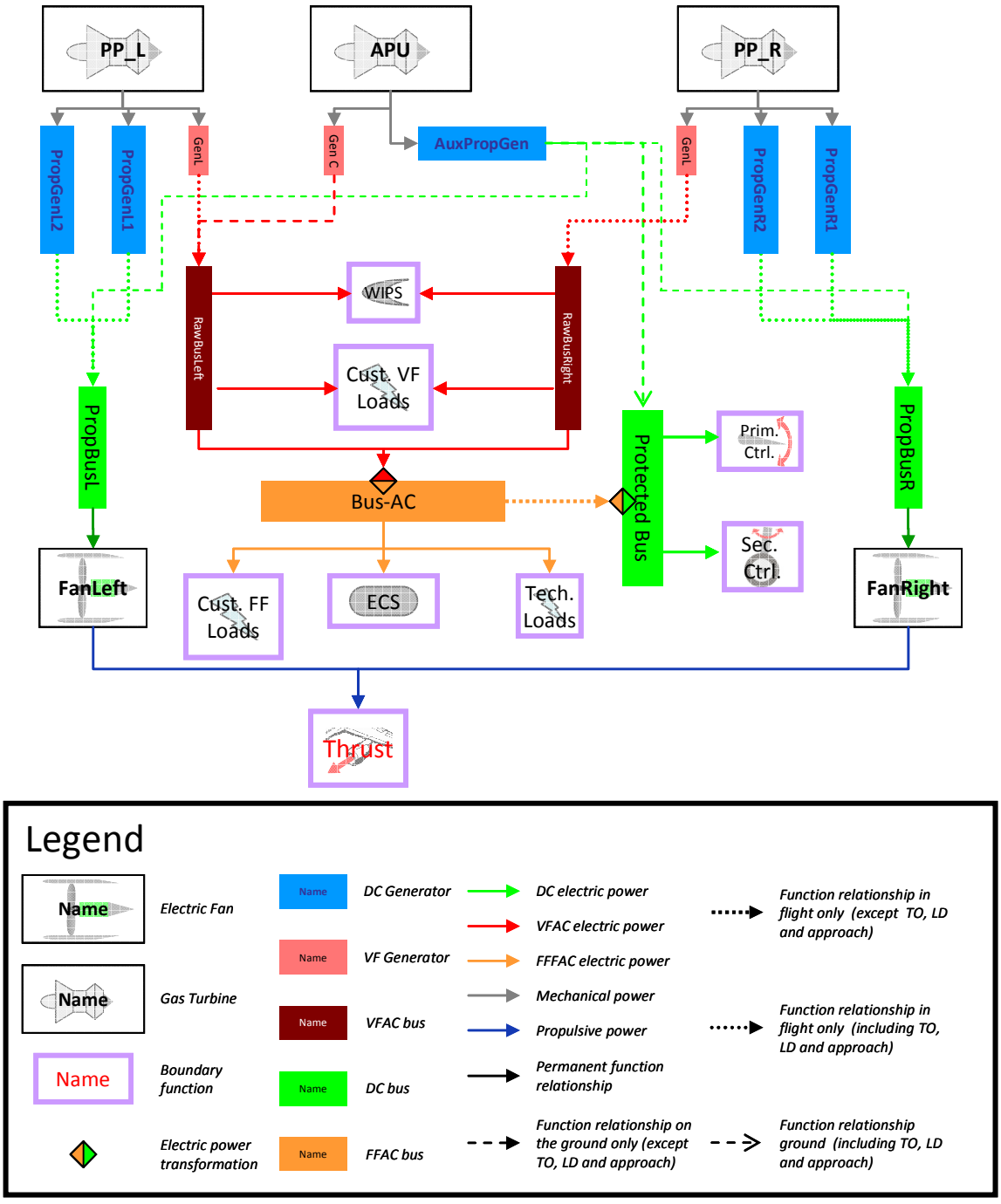


Figure 198: Overview of architecture 1 without failures

In order to get a better understanding of the four possible configurations which can be used in normal operation a detailed description of each is provided in the following figures

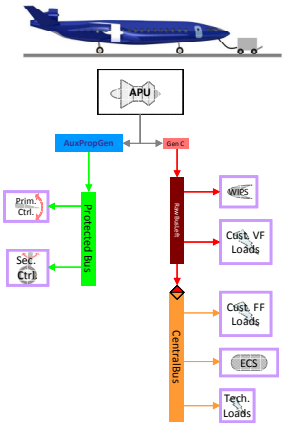


Figure 199: Arch.1 - Park configuration

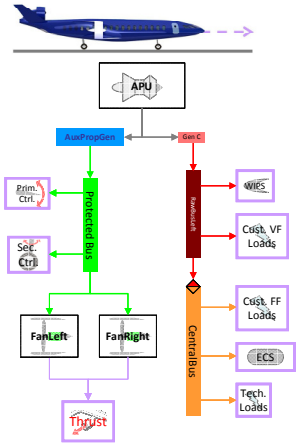


Figure 200: Arch.1 - Taxi Configuration

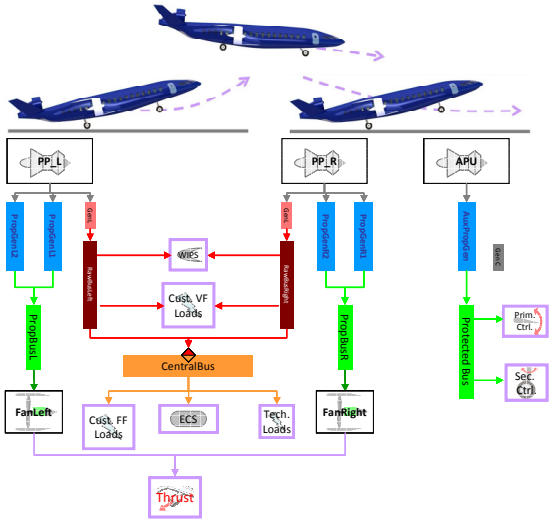


Figure 201: Arch.1 - Take-off configuration

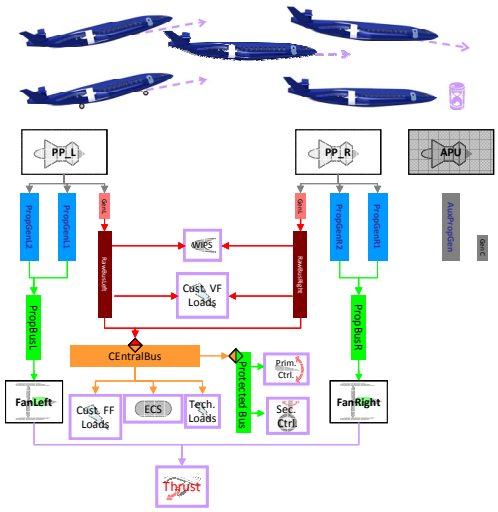


Figure 202: Arch.1 - Airborne configuration

In this architecture, several elements are expected to be identical. Therefore even if the requirements imposed upon them are different, their designs are expected to the same. These subsystems are defined as **symmetric**. In this architecture, the symmetric sets are formed by the following subsystems:

- The power plants are expected to be identical (except *APU*)
- The propulsive generators are all supposed to be the same (*PropGenL1*, *PropGenL2*, *PropGenL3*, *PropGenL4*). Note the sizing of the auxiliary propulsive generator can be different.

- The VF generators on the power plants (except *APU*) are expected to be identical (*GenL* and *GenR*).
- The electric fans are identical (*FanLeft*, *FanRight*)

6.3.1.2 Failure configurations

In order to simplify the definition of the architecture concept, only two types of failure scenarios will be considered. The first type is a dispatch failure, while the other will be a major failure (which implies that the original flight plan must be aborted). These failures will concern the VF and propulsive generators connected to the main power plant.

The first type of failure implies that one VF generator is out of order along with one propulsive generator. The subsystems must be sized in a such a way that any such combination of failures will still allow for operating in a dispatch mode. Since there are four propulsive generators and two VF generators, this failure can happen in any of the eight possible failure configurations presented by Table 33.

Table 33: Dispatch Failure configurations

Conf.	VFGen	PropGen		VFGen	PropGen		
Name	GenL	L1	L2	GenR	R1	R2	
1A	failed	OK	failed	OK	OK	OK	1a
1B	failed	failed	OK	OK	OK	OK	1b
1C	OK	OK	failed	failed	OK	OK	
1D	OK	failed	OK	failed	OK	OK	
1E	failed	OK	OK	OK	OK	failed	
1F	failed	OK	OK	OK	failed	OK	
1G	OK	OK	OK	failed	OK	failed	1c
1H	OK	OK	OK	failed	failed	OK	1d

The failure configurations listed above will directly influence the size of six subsystems: *GenL*, *GenR*, *PropGenL1*, *PropGenL2*, *PropGenR1* and *PropGenR2*. In order to decrease the number of scenarios to implement, the role of each subsystem for all failure configurations should be considered. The effect on the generators is presented in Table 34 and Table 35. The percentages indicate its share in producing the overall propulsive or non-propulsive power.

Table 34: Effects of dispatch failure configuration on non-propulsive generators

Conf.	PropGen	
Name	GenL	GenR
1A	0%	100%
1B	0%	100%
1C	100%	0%
1D	100%	0%
1E	0%	100%
1F	0%	100%
1G	100%	0%
1H	100%	0%

Table 35: Effects of dispatch failure configurations on propulsive generators

Conf.	PropGen			
Name	L1	L2	R1	R2
1A	100%	0%	50%	50%
1B	0%	100%	50%	50%
1C	100%	0%	50%	50%
1D	0%	100%	50%	50%
1E	50%	50%	100%	0%
1F	50%	50%	0%	100%
1G	50%	50%	100%	0%
1H	50%	50%	0%	100%

If we consider these tables we see that, for the propulsive generators, the failure configurations are identical in pairs. For the non-propulsive generators the experiments are going to impose two fundamentally different sizing constraints. These observations can be summarized by the following statements: For the propulsive generators the independent sizing constraints are captured by four of the failure configurations: A or C, B or D, E or G and F or H. For the non-propulsive generators, the minimal set is constituted by: A or B or E or F and C or D or G or H. Based on these minimal set we can say that the sizing constraints will still be captured if we were only to consider the four configurations indicated on the right of Table 33.

The second type of failure corresponds to a situation where one propulsive generator on each side has been lost along with one non-propulsive generator. Based on this description eight possible failure configurations could be observed. These configurations are listed in Table 36.

Table 36: Major Failure configurations

Conf.	VFGen	PropGen		VFGen	PropGen		
Name	GenL	L1	L2	GenR	R1	R2	
2A	failed	failed	OK	OK	failed	OK	1a
2B	failed	OK	failed	OK	failed	OK	
2C	failed	failed	OK	OK	OK	failed	1b
2D	failed	OK	failed	OK	OK	failed	
2E	OK	failed	OK	failed	failed	OK	1c
2F	OK	failed	OK	failed	OK	failed	
2G	OK	OK	failed	failed	failed	OK	1d
2H	OK	OK	failed	failed	OK	failed	

Based on a similar analysis to the one presented earlier, the sizing constraints can be summarized by the subset of failure configurations indicated on the right of Table 36. Overall 8 degraded states must be captured in the architecture analysis. In order to do so, it is necessary to represent the structure associated with each failure. The following figure provides an overview of the architecture configurations associated with each failure.

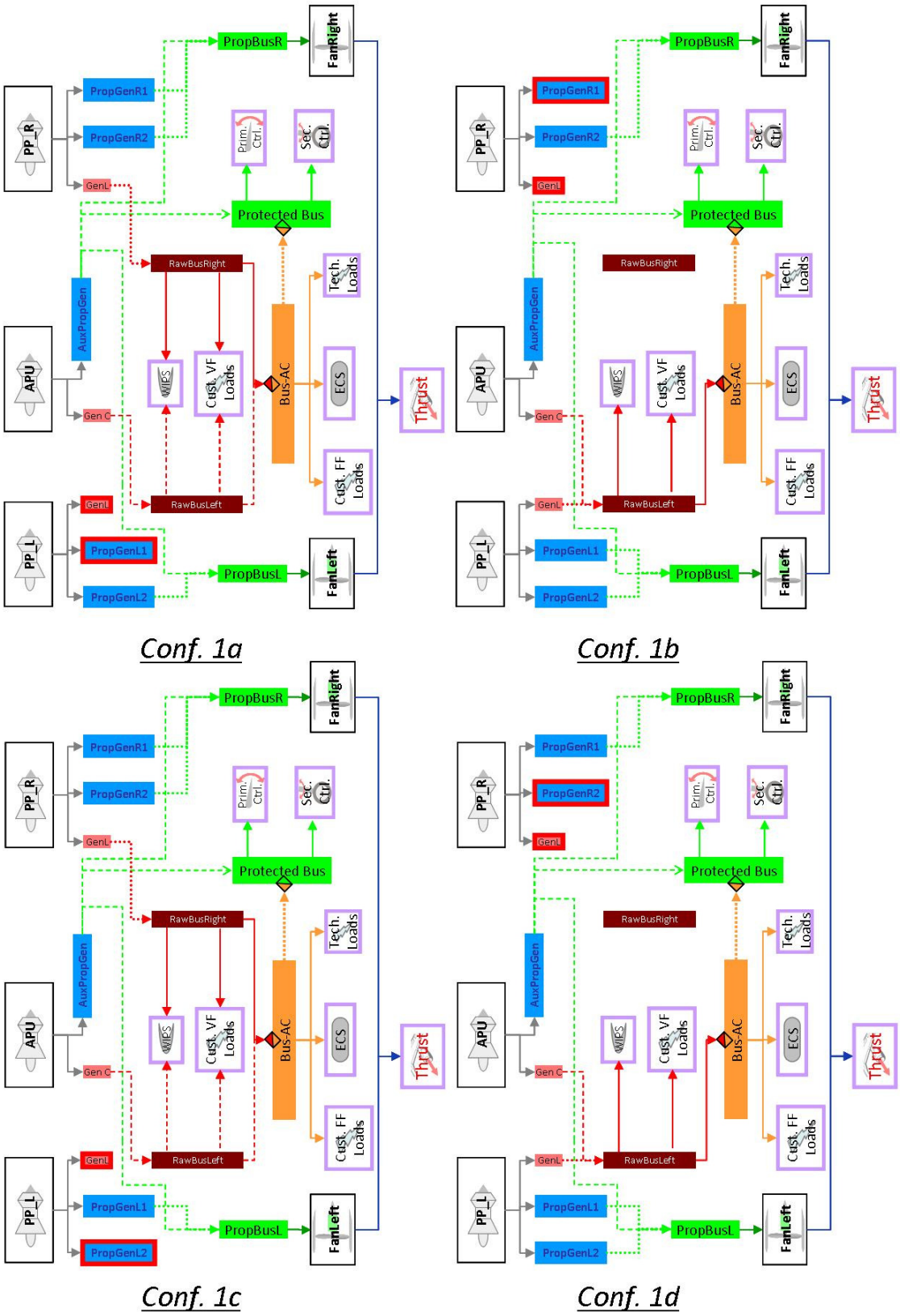


Figure 203: Overview of configurations associated with failure 1

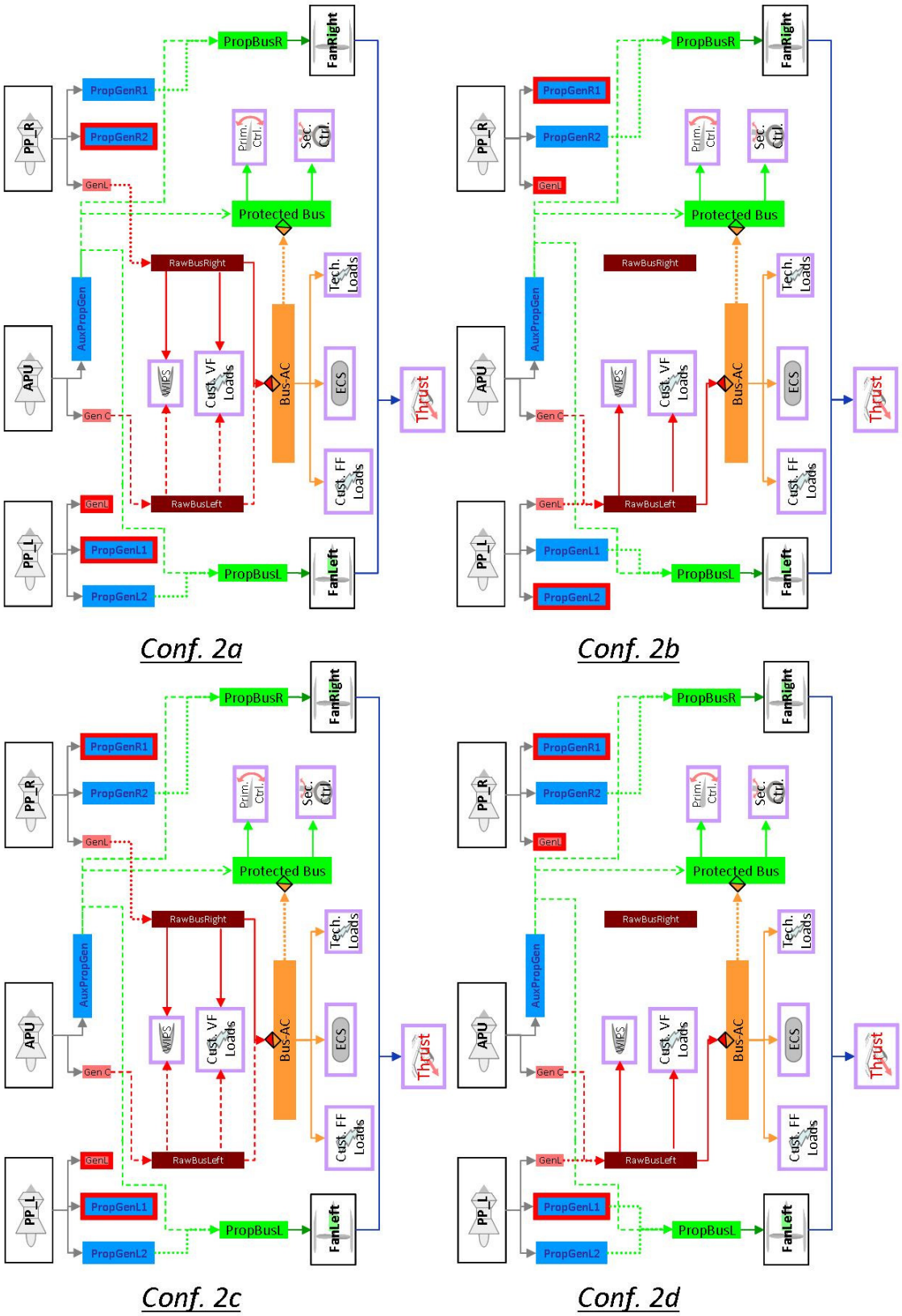


Figure 204: Overview of configurations associated with failure 2

6.3.2 **Architecture Concept Definition in SysML**

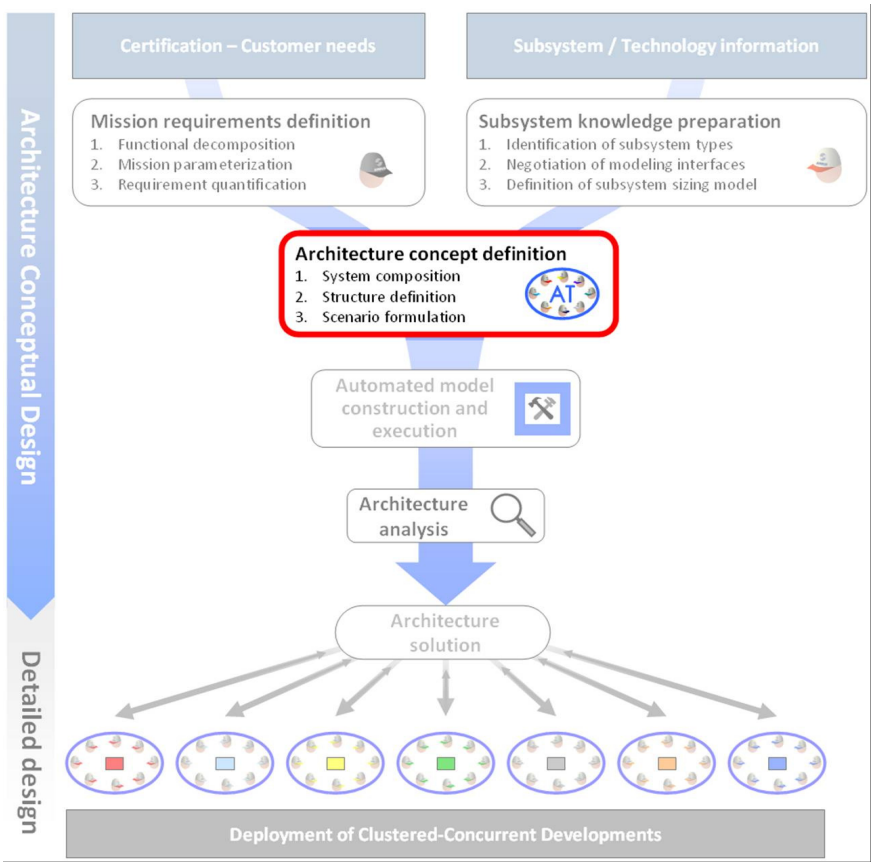


Figure 205: Process Overview – Architecture Concept Definition

This activity is performed collaboratively by the architecting team. The objective is to define the concept with SysML graphs. This description is based on the Subsystem Concept Library defining the subsystem and function concepts. A notional overview of the process in its implementation is presented in Figure 206.

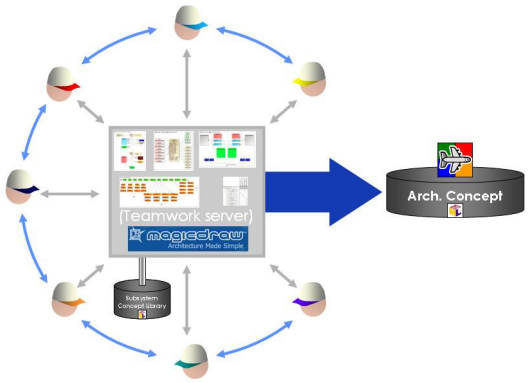


Figure 206: Overview of the Architecture Concept Definition activity implementation

The outcome of this activity is the SysML description of an architecture concept. The conceptual model representing it is represented on the right hand side of Figure 206. The architecture concept definition is implemented in the MagicDraw environment. A generic description of the process associated with the definition of architecture concepts is presented in Appendix J which presents a User Guide for using MagicDraw in the definition of the architecture concept. This step is performed collaboratively by the architecting team. In the test case presented in this thesis, the architecture concept was declared by a team composed of Charles Nespoulous[4] and the author. This team definition approach was used to test the ability to perform the tasks collaboratively in a context as similar as possible to the one corresponding to a design office.

The architecture definition process, presented in this section, describes parameter X_{arch} in the mathematical formulation presented earlier. This parameter will define the elements and structure of the analysis model described by the optimization problem presented in equation (53) represented below.

$$\underset{\Gamma_\bullet}{Max} \quad OpRange\left(W_\bullet, \overline{FB}_\bullet\right) \tag{53}$$

Subject to:

- $\left[\overline{R}_\bullet, \overline{Spe}\right] = Structure\left(\overline{R}_{mission}, \overline{Spe}_{mission}, X_{arch}, \overline{IndFctReq}_\bullet\right)$
- $\left[W_n, \overline{FB}_n, \overline{IndFctReq}_n\right] = SubsysSizing\left(\Gamma_n, \overline{R}_n, \overline{Spe}_n\right)$ for $n \in [1, N]$

[4] Graduate research assistant at the Aerospace System Design Laboratory

6.3.2.1 Definition of the Architecture Concept

In the architecture concept definition process, the first graph to be implemented is the composition graph. This graph lists the subsystem elements composing the architecture. The composition graph, implemented in a block definition diagram (SysML), is presented in Figure 207. Note that the colors indicated in the graph were added to highlight the type of the subsystem declared in this graph. The colors are not expected to carry any particular SysML meaning in the context of this graph. We can see that the graph presented above makes use of the subsystem blocks prepared in the "Identification of Subsystem Types" phase. These blocks, stored in the "Subsystem Concept Library", contain (or refer to) all the information necessary for the builder to integrate and associate the modeling bricks necessary to the analysis of the architecture concept.

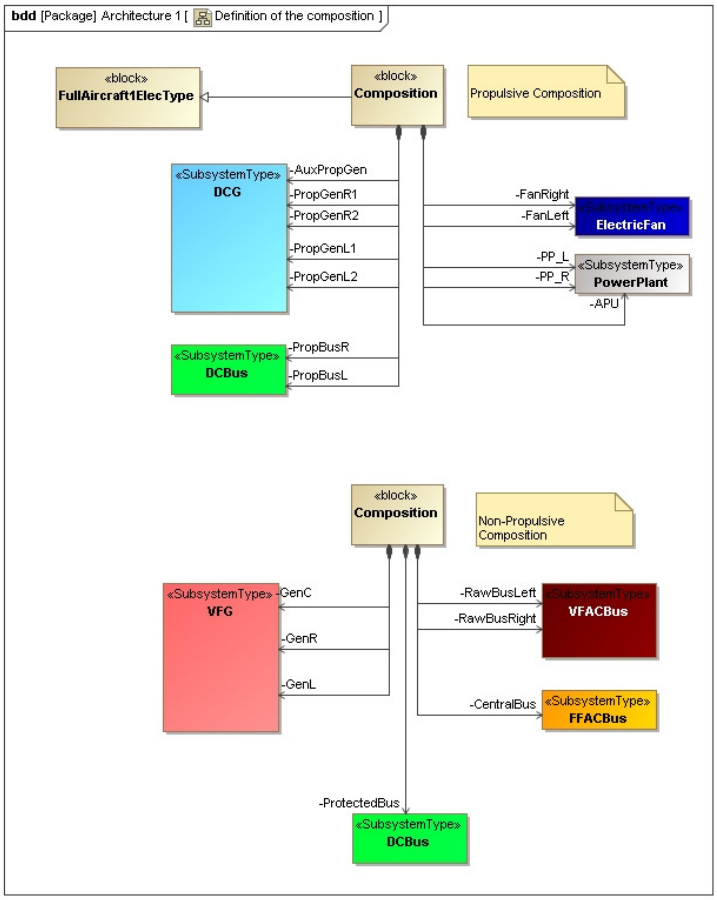


Figure 207: Definition of Architecture 1 composition

The architecture structure definition was more involved modeling-wise. This volume of workload associated with its definition results from the number of configurations constituting the structure and the number of elements composing the architecture. The following graph presents an overview of the configurations considered in the sizing of the architecture. These configurations are listed and related to the block presenting the architecture composition by the specialization relationship required in the methodology.

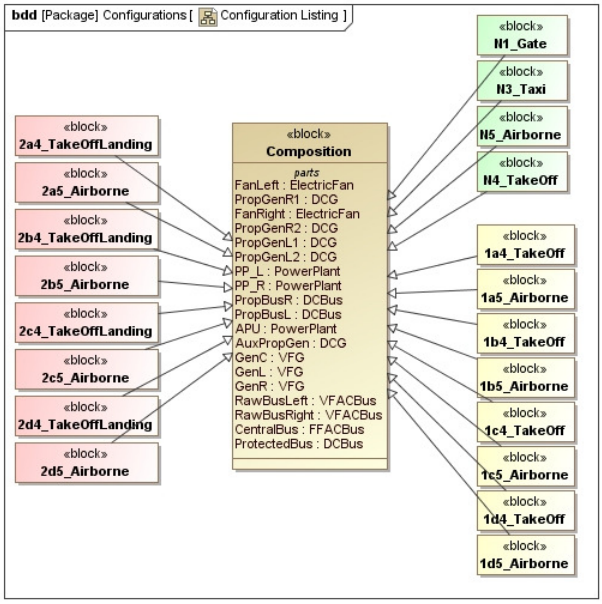


Figure 208: Listing of architecture configurations

In normal operations the architecture is susceptible to operate in four possible configurations: park, taxi, take-off (and landing) and airborne. In addition to normal configurations, 8 failure types were defined previously (4 for the double generator failures and another 4 for the triple generator failure). In each failure type, the architecture may be operating in two possible configurations, the take-off (and landing) configuration and the airborne configuration. Overall twenty configurations needed to be defined (4 for normal operations plus 2 times 8 for failed configurations).

The architecture is composed of 19 subsystems performing 8 boundary functions. In order to make sure that no information was misrepresented or forgotten, the definition

of each configuration was performed based on two internal block diagrams (with the exception of the park and taxi configurations which were simple enough to be represented in the same graph). The thirty-two graphs will not be represented in this description. Instead a sample of the key figures is presented in the following pages. The selected graphs were those representing the architecture composition at park and take-off. For each failure type only the take-off presentation will be shown to highlight the principle used in the formulation of the configurations.

In the graphs presented in the following pages the colors on the boxes represent the type of the subsystem. The ports have been colored by the type of function the supply or require. These colors do not carry any particular SysML meaning. Therefore, these colors were not necessary (from a semantic point of view) but they helped with the visual navigability of the graphs (which otherwise may have been clogged up by text indicating the types of the port and subsystems).

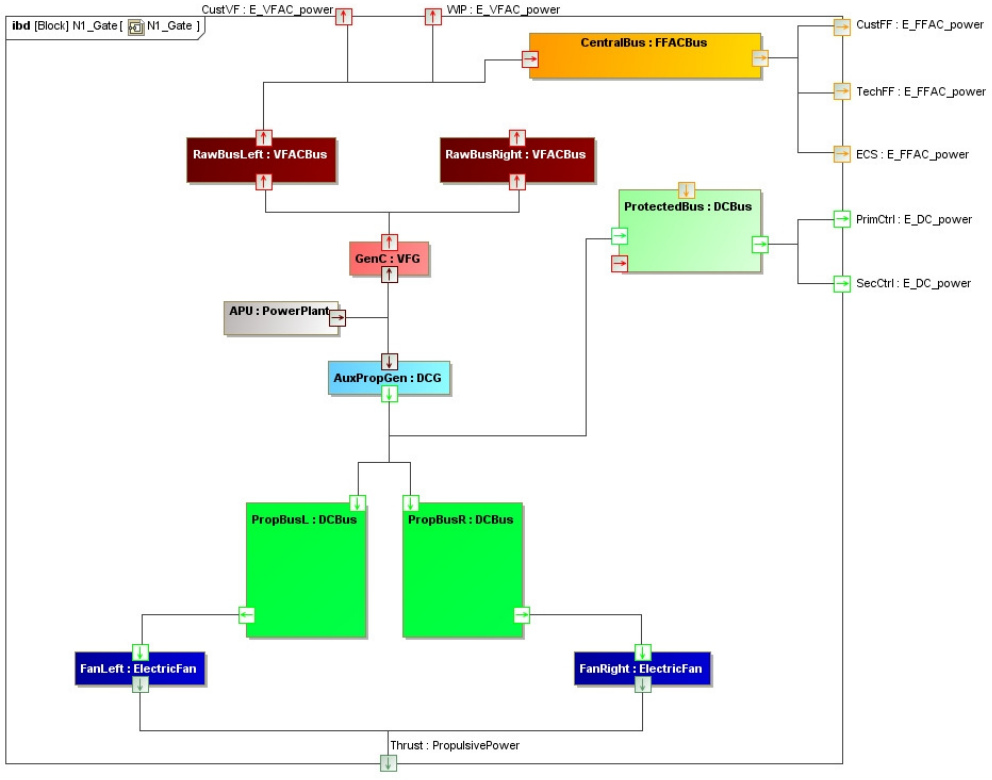


Figure 209: Park configuration - Architecture 1

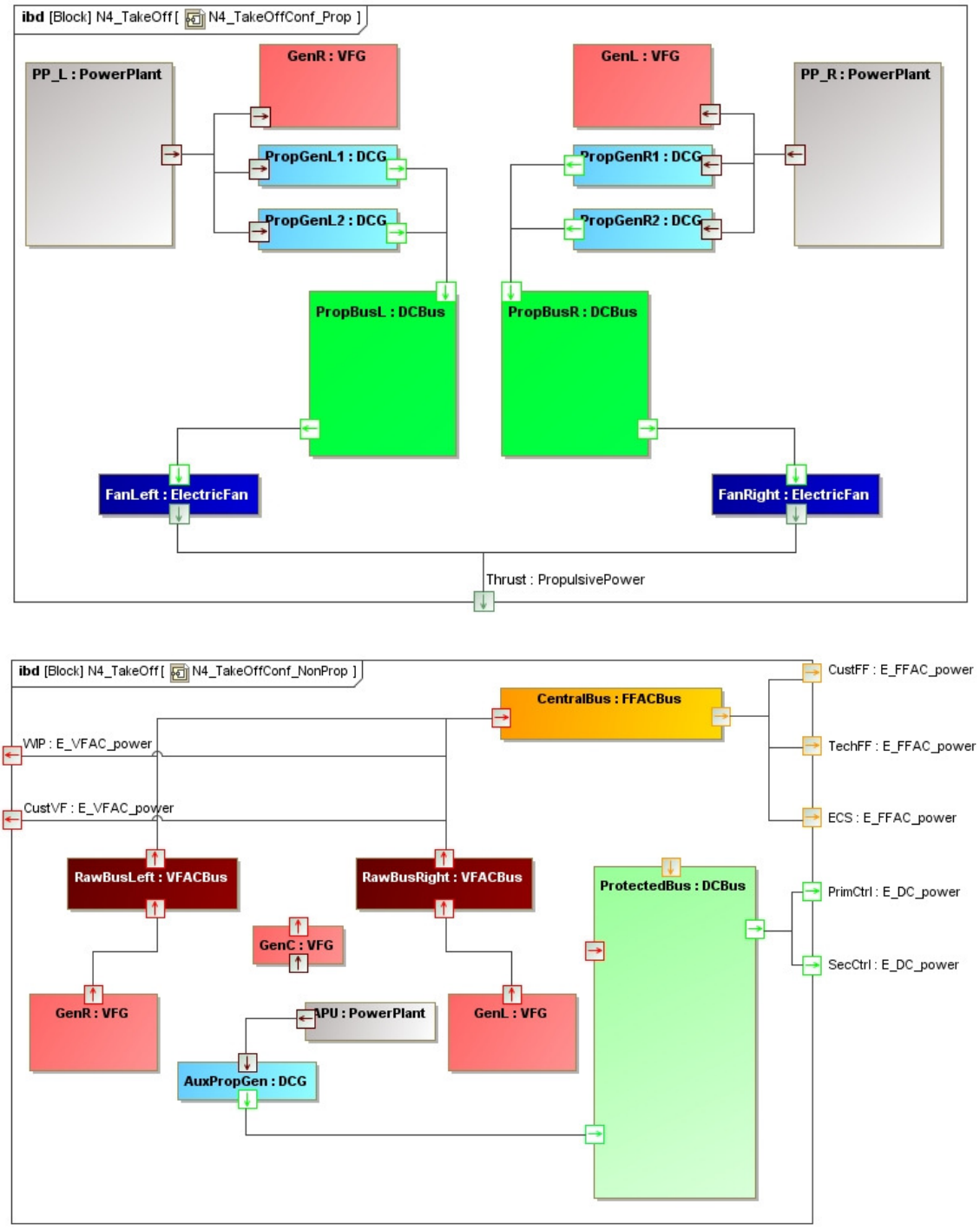
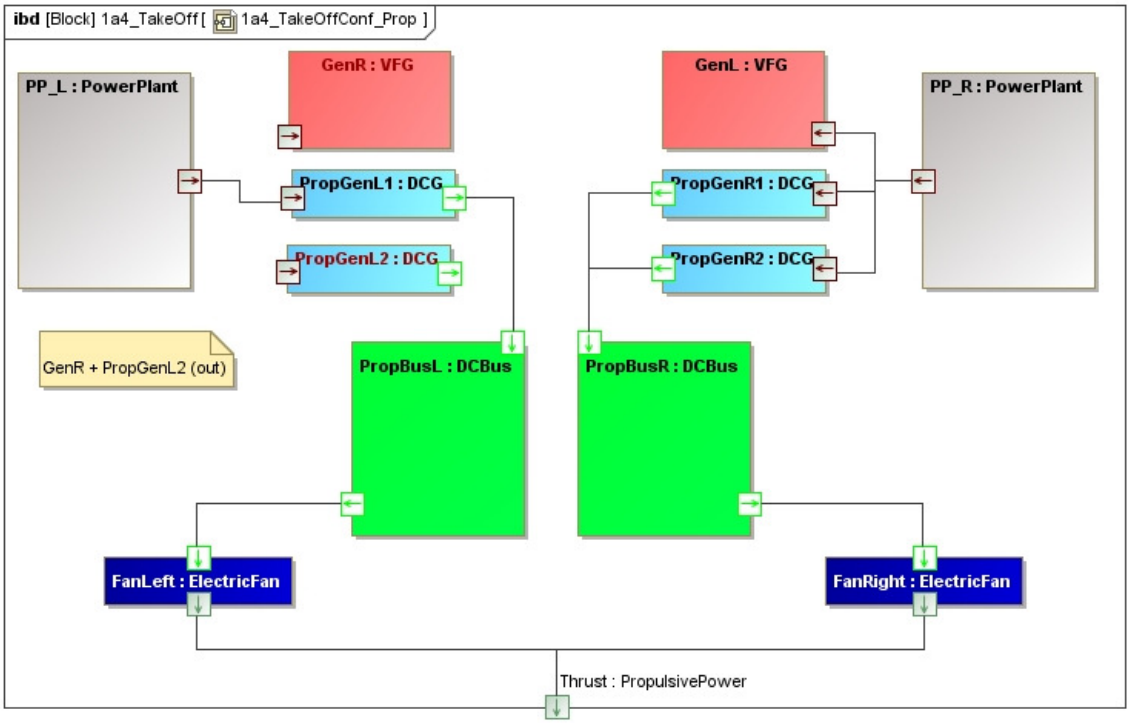


Figure 210: Normal operation take-off (and landing) configuration - Architecture 1

It is important to note that the implementation of new configurations does not require the redefinition of all the connections. A simple copy paste in MagicDraw is necessary to reproduce a configuration. Based on this reproduction, the architects only

need to remove or add the connections differentiating the new configuration from the original. In order to illustrate this principle, the configurations associated with the take-off with failed subsystems are presented in the following figures.



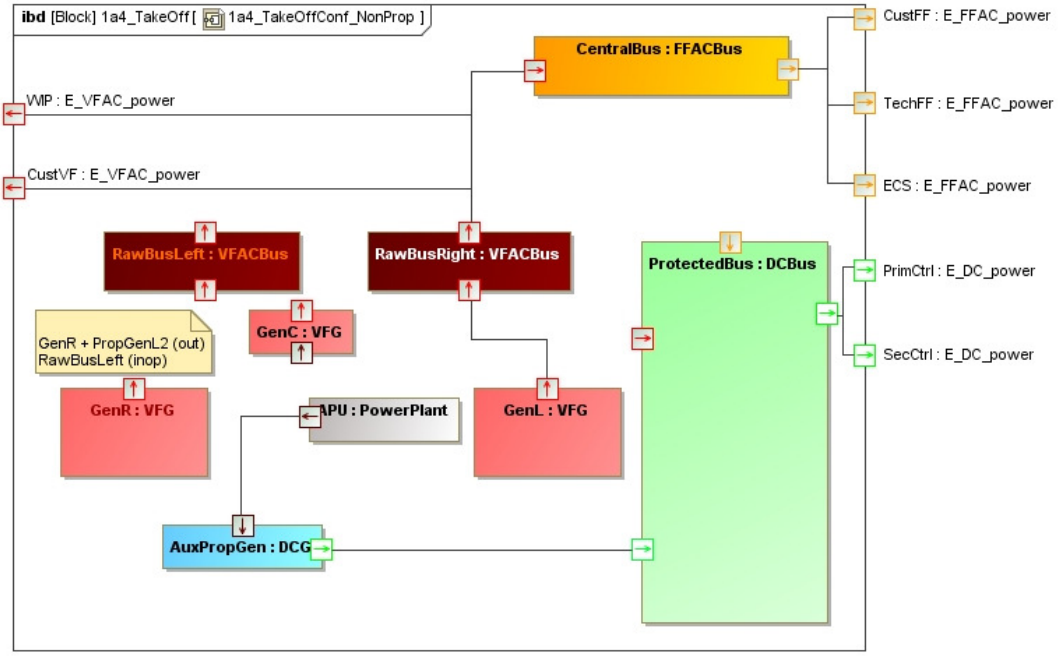


Figure 211: Failure 1a take-off (and landing) configuration - Architecture 1

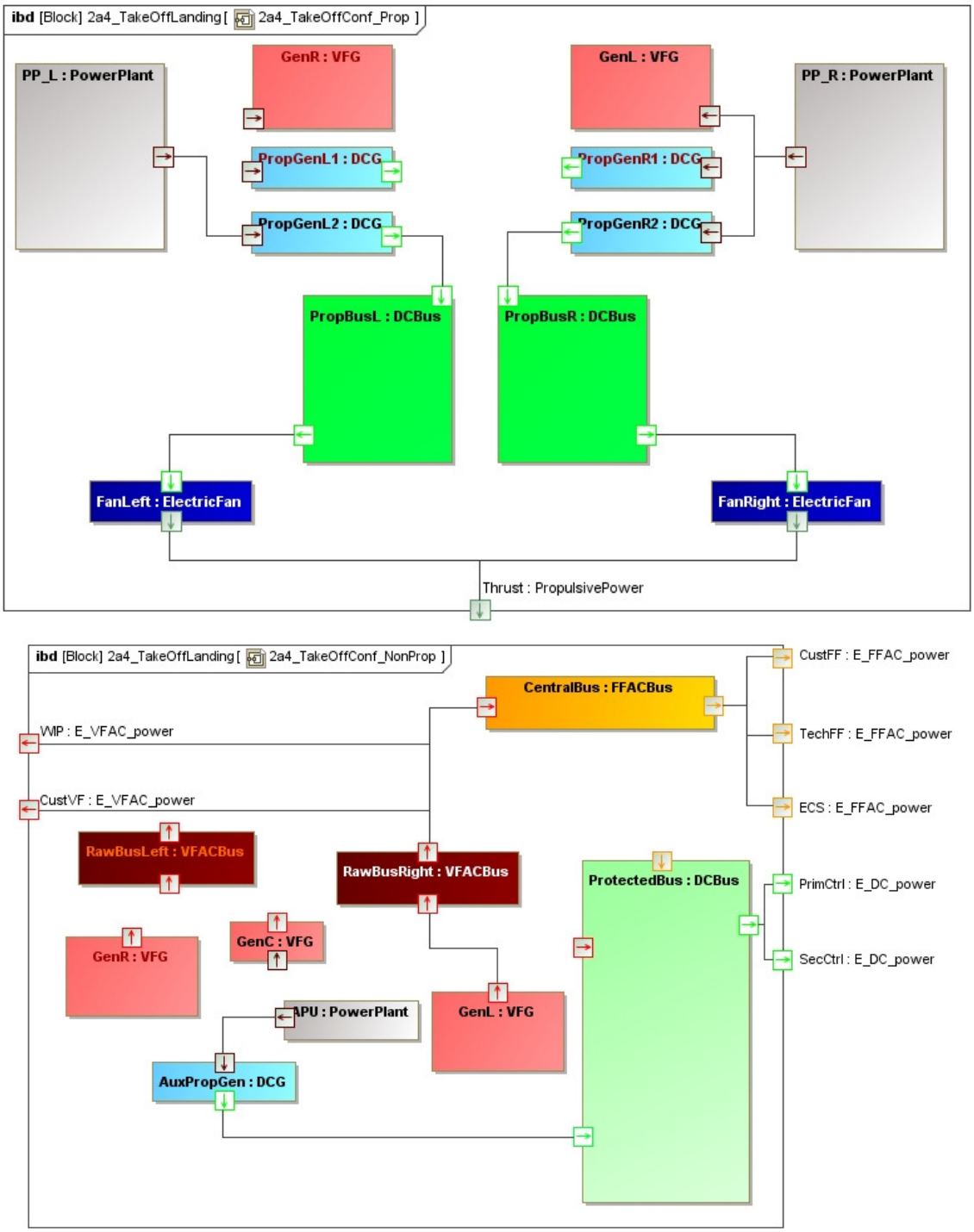


Figure 212: Failure 2a take-off (and landing) configuration - Architecture 1

Based on the mission prescription on sequences of flight, the operational sequences were described by two graphs. The first describes the dispatch failure sequences while the second presents the major failure sequences. The failed sequences are presented on a chronology supported by the normal mission sequence. In order to facilitate the readability of the graph color codes have been used. The green nodes represent the normal operation scenarios, the yellow and orange nodes the dispatch and major failure scenarios. Two graphs representing the possible mission sequences are presented in the following figure.

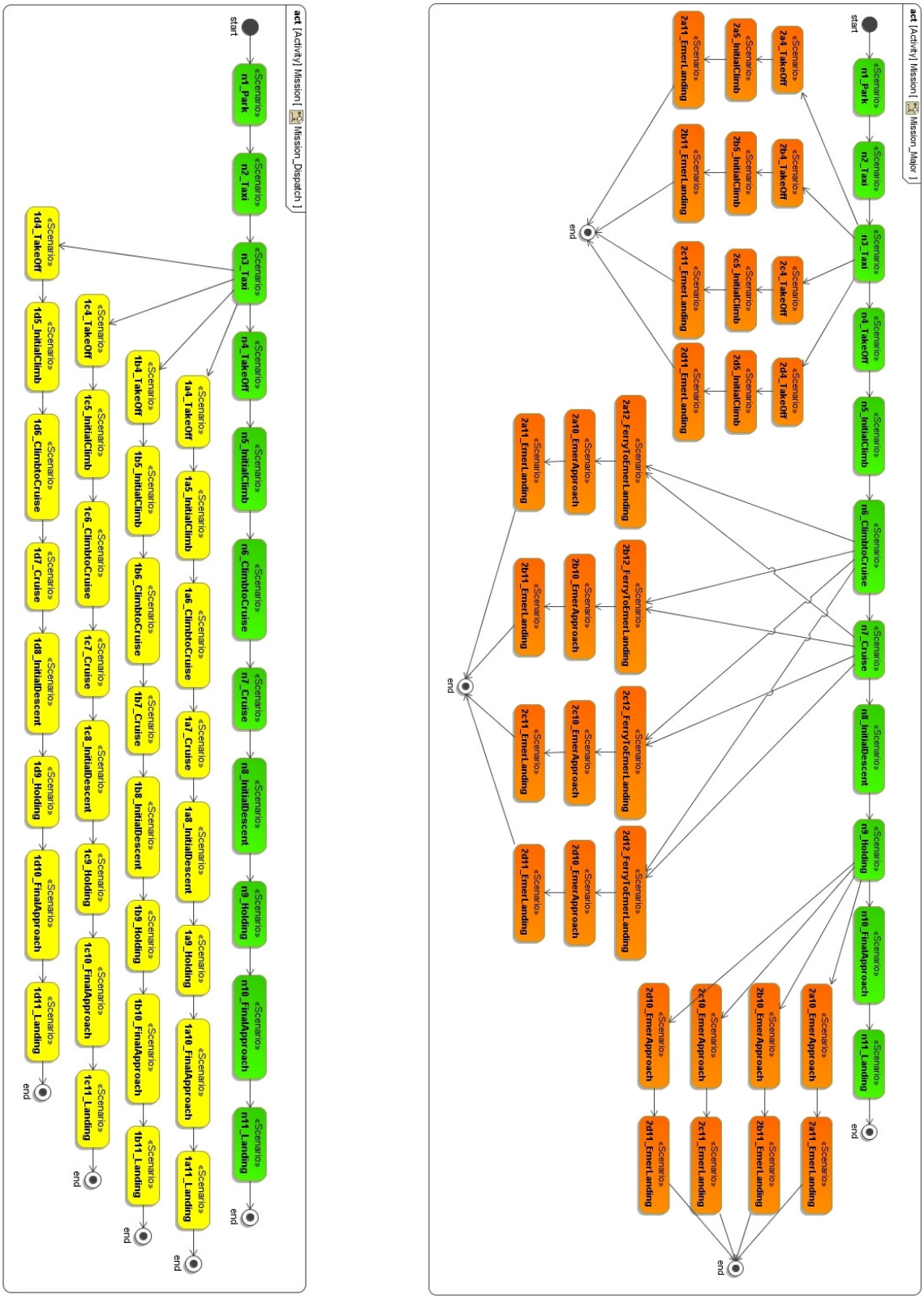


Figure 213: Mission sequences - Architecture 1

In order to allocate each operational scenario to an architecture configuration, the following 3 allocation matrices were used. Each matrix allocates the scenarios with the same degree of criticality (normal, dispatch failures and major failure).

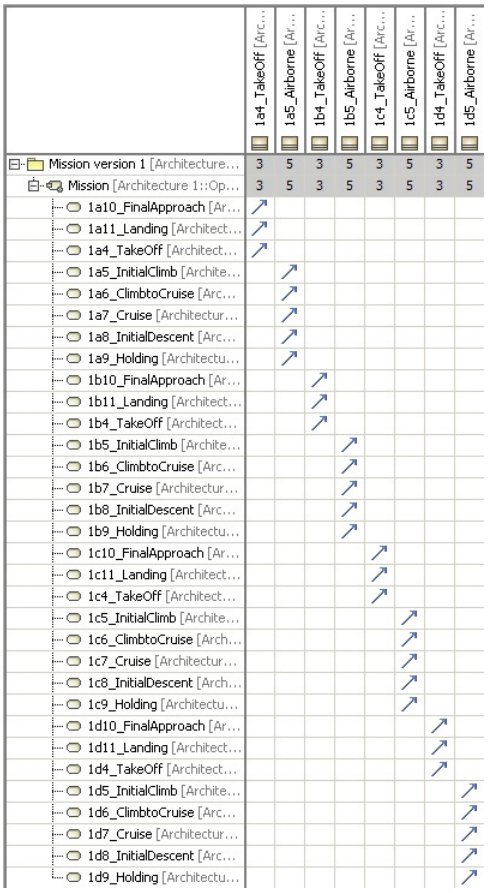


Figure 214: Allocation Matrix for dispatch failures

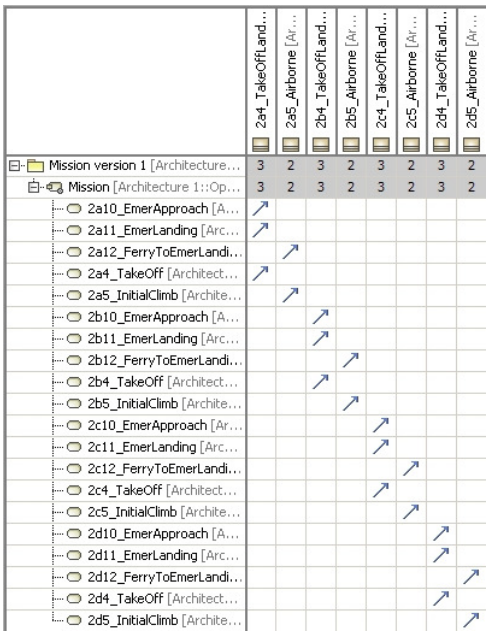


Figure 215: Allocation Matrix for major failures

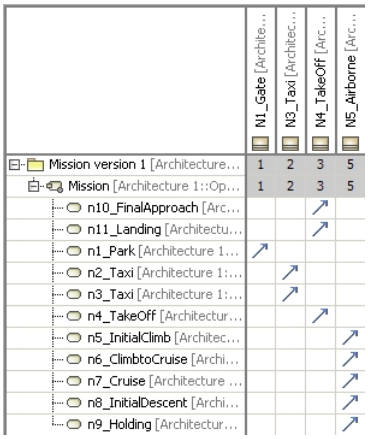


Figure 216: Allocation Matrix for normal operations

6.3.2.2 Observations on Workload Implied by the Architecture Definition

Overall, 45 graphs were defined collaboratively using the "Teamwork server" on MagicDraw. The implementation of this model took about 4 human work hours. The breakdown of the time dedicated each of the tasks is presented in the table below:

Table 37: Time cost of the architecture concept definition

		Duration
Composition		30 min
Structure	Normal	60 min
	Dispatch	45 min
	Major	45 min
Op.	Sequences	40 min
	Conf. allocation	20 min
Total		4 hours

Using the teamwork server, these tasks could be parallelized. Since in this example the "team" was only constituted of two members, the parallelization rates of the tasks were fairly limited. The following PERT chart decomposed the architecture definition process. It highlights the opportunity to parallelize the architecture concept definition activities. The definition of the failure configurations can be performed in parallel. The definition of the mission sequences does not necessitate the failure configurations to be ready. A convergence point is imposed by the allocation of the scenarios to the configuration. Therefore, using the MagicDraw Teamworker server, the architecture concept could very well be defined in little above two and a half hours.

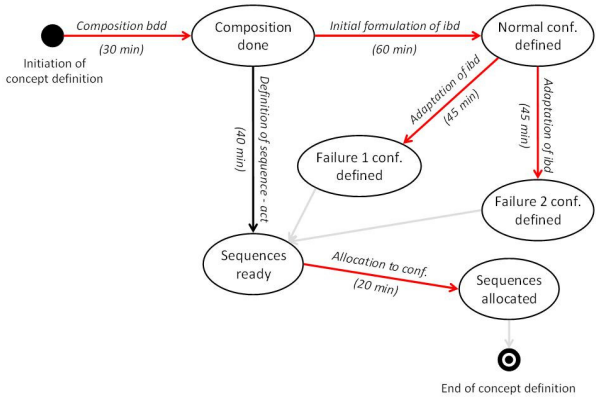


Figure 217: PERT chart of the architecture concept definition

6.3.2.3 Conclusions on Architecture Concept Definition

The implementation of the methodology has shown that using the SysML language, in the way prescribed by the methodology, enables capturing complex architecture concepts. This example has shown both by the flexibility of the language and the precision of its semantics.

The visual advantages offered by SysML enabled the implementation of the method at minimal implementation cost. MagicDraw was a tool developed long before the formulation of this thesis (therefore it was not built for supporting the methods proposed in here). Nevertheless, via its SysML feature, MagicDraw offered a professional, robust and visual interface for the architecture developments which could easily (at minimum implementation cost) be integrated in the framework.

The great flexibility offered by SysML also induces an important challenge. The great number of possible ways to represent the information in SysML increases the complexity of the learning process of the language and the intellectual effort necessary to understand the architecture concept implemented. In order to address this challenge, a strict process is proposed for the implementation of the architecture concept definition. The concept definition method is based on a somewhat simple, but flexible, breakdown of the tasks, where each task is associated with specific SysML graphs. Overall only four types of graphs were used in the definition of the architecture concept. The definition process described in the User Guide (provided in Appendix J) provides a simple and systematic way to describe the architecture concept. Following this systematic process structures the information which simplifies both the definition process and the interpretation of the model. This deduction is confirmed by observations made on Georgia Tech students who have used the methodology. Their learning process did not exceed a few weeks.

Another minor challenge imposed by SysML was the disconnect between the information displayed on the graphs and the actual information stored in the model. The

definition of graphs implements the information in the SysML model. If this graph is erased, the information it has defined is still in the model even if the information is not graphically represented anymore. Therefore, what you see on the graph does not provide you with a complete view of the model. This disconnect does offer important advantages. For instance, the definition of configurations was too complex to be defined on a graph single in a practical fashion. Using two graphs allowed the user to declare the large and complex web of relationships implied by this configuration with a practical visual simplicity.

It can be concluded that SysML offers effective visual means and a robust environment for the definition of architecture concepts. Since SysML has a wider scope than what was necessary for the implementation of the methodology, it offers more features than what was strictly necessary to do the task at hand. This abundance of features presented superficial complexity in the implementation. This complexity was palliated by the conciseness and simplicity of the methodology and the definition of a user guide which provides guidelines for the definition of the architecture concept.

6.4 Description of the Builder

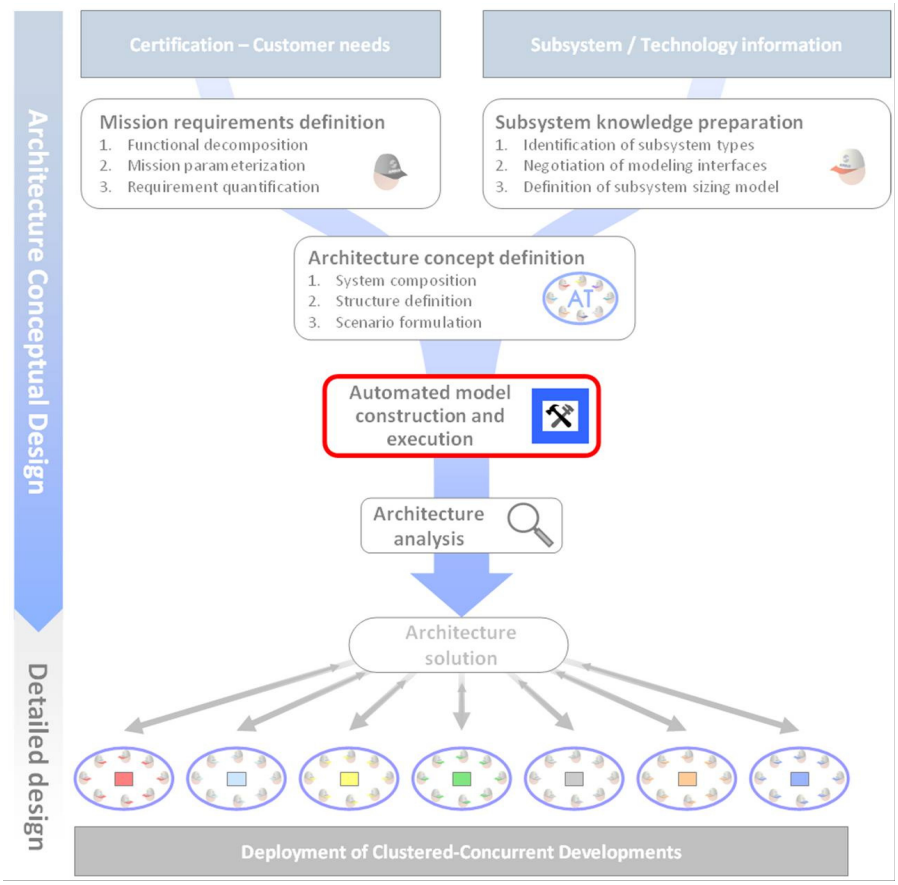


Figure 218: Process Overview – Automated construction of the analysis model

The construction of the model is performed by the Architecture Analysis Builder (or Builder). This element is implemented by a Java routine which translates the SysML model (conceptual model) into the ModelCenter model (or analysis model). This section will provide an overview of the implementation of the Builder. This overview will first remind the reader of the place of the builder in the overall methodology by presenting its conceptual role. Then an overview of the code is presented in the second subsection. This overview will present the general structure of the builder code (which is presented in its entirety in Appendix I). This section will be concluded by general observations on the implementation of the builder.

6.4.1 **Conceptual Role of the Builder**

The role of the Builder is to construct the ModelCenter analysis corresponding to the architecture concept previously defined in SysML. As described in the methodology chapter, it does so by assembling the modeling bricks previously defined in the subsystem knowledge preparation and stored in the Analysis Server library. An overview of this role is provided in Figure 219.

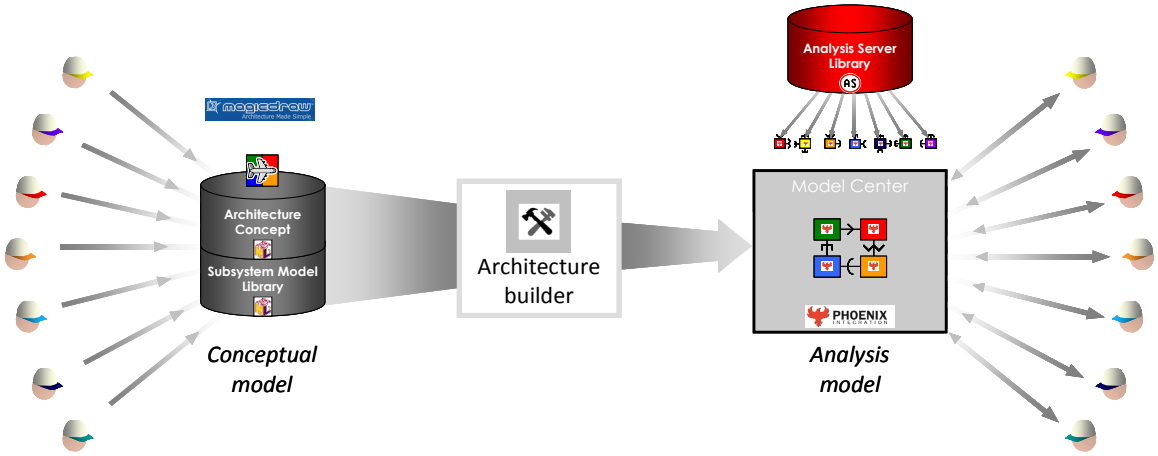


Figure 219: Overview of the architecture analysis builder implementation

From a conceptual standpoint, the action of the Builder can be seen as the element constructing the optimization problem constituting the architecture analysis (represented in equation (53) reproduced below).

$$\underset{\Gamma_\bullet}{Max} \; OpRange\left(W_\bullet, \overline{FB}_\bullet\right) \tag{53}$$

Subject to:

- $\left[\overline{R}_\bullet, \overline{Spe}\right] = Structure\left(\overline{R}_{mission}, \overline{Spe}_{mission}, X_{arch}, \overline{IndFctReq}_\bullet\right)$
- $\left[W_n, \overline{FB}_n, \overline{IndFctReq}_n\right] = SubsysSizing\left(\Gamma_n, \overline{R}_n, \overline{Spe}_n\right)$ for $n \in [1, N]$

Based on the expression of X_{arch} (i.e. the conceptual model describing the architecture), the builder is going to import the modeling bricks representing the N subsystems constituting the architecture. It will also generate the connections necessary to implement the mutual constraints between subsystems. These mutual constraints or

Structure() transmit the requirements implied by functional relationships between subsystems. If two subsystems are inter-related, it will take the induced requirement from one subsystem to size the other. If a subsystem is performing a boundary function, it will use the mission scenario definition to quantify the requirement necessary to its sizing.

6.4.2 **Implementation of the Routine**

The Architecture Builder is implemented by a Java routine stored as a MagicDraw plug-in. The coding of the Java routine was performed in the Eclipse environment [86]. In this context, the Eclipse routine was used to debug the code and compile it into a Java Archive (JAR) file. The code is composed of seven classes represented in their tree structure in Figure 220.

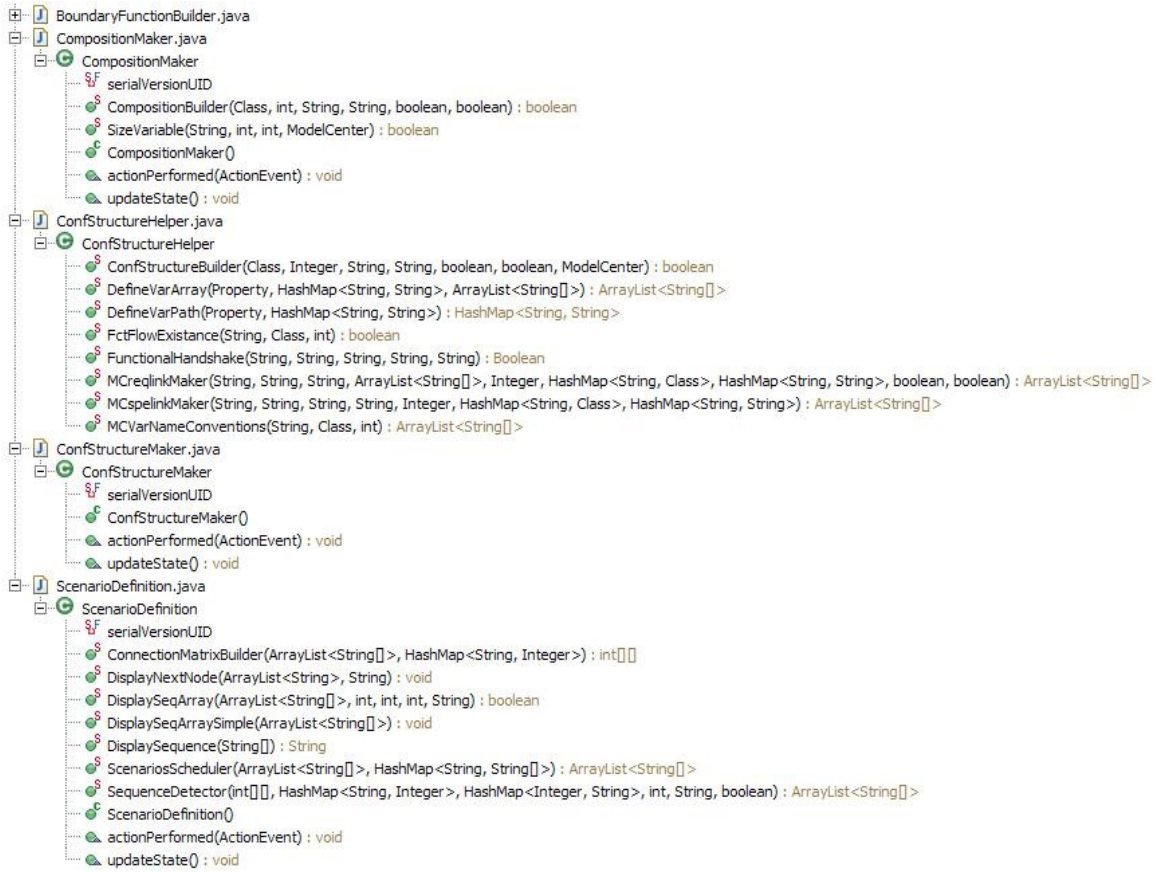


Figure 220: Overview of builder classes

The code used to implement these Java classes is presented in appendix I. The final JAR file was stored under the MagicDraw plug-in folder. Using this plug-in the MagicDraw environment was augmented by a new action group presented in Figure 221.

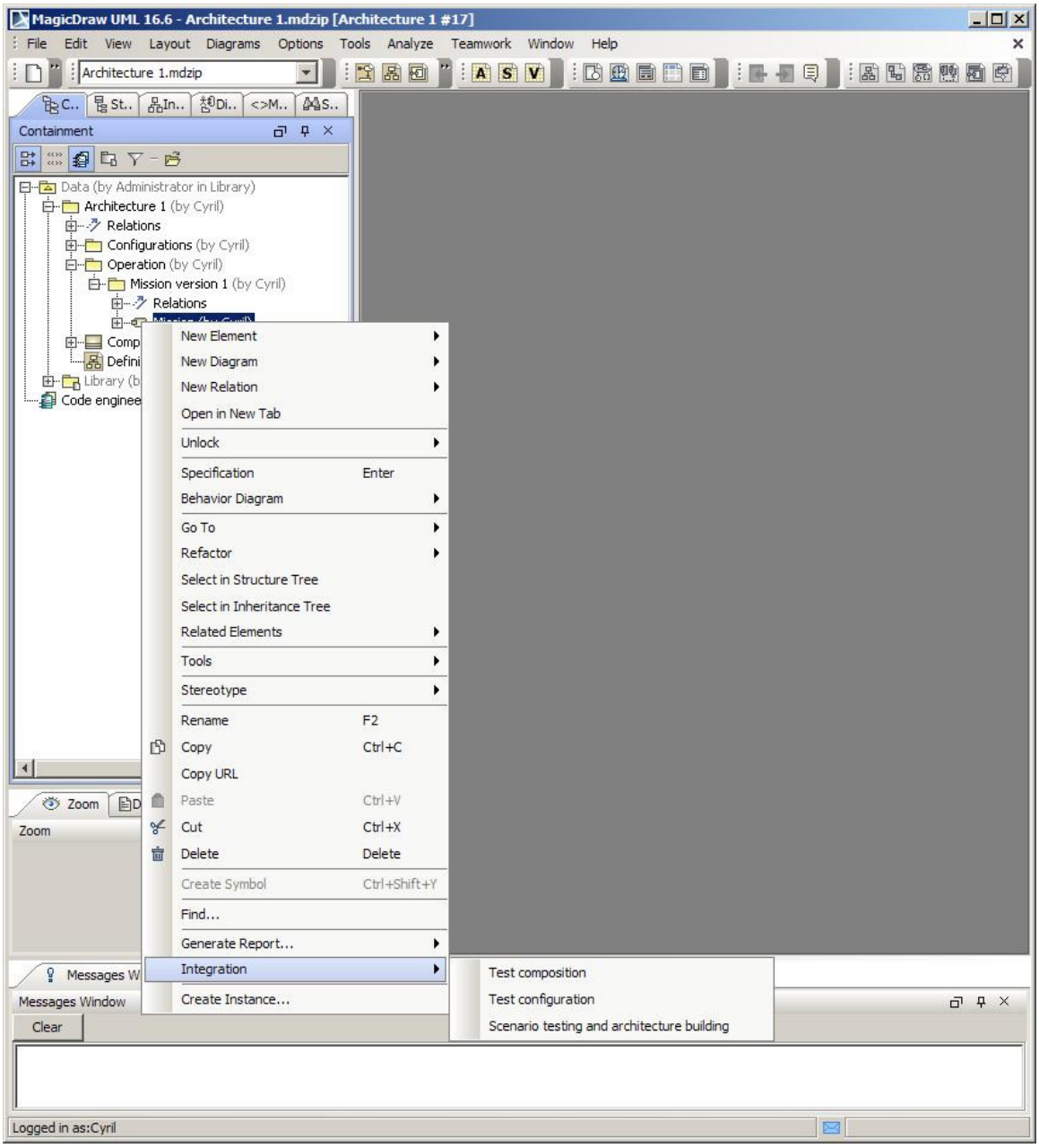


Figure 221: Builder action group in MagicDraw

The builder action group available from the containment tree in MagicDraw, offers 3 options:

- Test composition: This action launches the Builder on the block describing the architecture composition. The Builder imports the modeling bricks and makes the basic connections necessary to the representation of the composition.
- Test configuration: This action constructs the connections associated with a selected configuration block. It links the modeling bricks in the way specified by the functional flows.
- Scenario testing and architecture building: This action provides two options. The first will only read the operational definition and will return to the screen the possible sequences detected in the diagrams. The second option will not only read the operational definition but it will continue with the construction of the final model.

The third action allows for the final construction of the analysis model. The others enable testing the different elements of the architecture concept definition. As discussed in the previous section, it is sometimes possible to make a mistake in formulation of the SysML model. Using the testing actions allows for pin-pointing of the misconception in the model. For more debugging information please refer to the user guide in Appendix J.

6.4.3 **Observations on the Builder Implementation**

This builder was implemented in the end of the fall semester 2009 over a period of time that did not exceed 2 months. The bulk of the task was performed by one person (the author whom had no previous Java experience) with the initial guidance of an expert (Alek Kerzhner from the Systems Realization Laboratory at Georgia Tech). Therefore even if the code presented in the appendix may seem fairly involved, the formulation of the overall logic was greatly simplified by the formulation of the methodology which preceded the implementation. It is important to observe that two months of programming (by a novice) can not be considered as a highly sophisticated software project. Therefore,

we can conclude that implementing the methods proposed in this thesis is not a complex task.

Despite its software simplicity, the Builder's abilities are significant. Based on a graphical definition of the architecture concept, it is able to produce conceptually accurate architecture analysis models. A description of the analysis model generated in this proof of concept is provided in the following section.

6.5 Architecture Analysis Overview

Based on the description of the architecture concept and the modeling bricks defined in earlier phases, the architecture builder has created an architecture analysis model automatically. The tasks performed by this analysis model are:

- To size the subsystems based on their role in the architecture
- To synthesize the subsystem attributes in order to estimate the performance of the architecture concept.

In order to see how the model performs these objectives this section analyzes it under its different aspects. First it will observe and describe the generic elements of the model. These generic elements provide the backbone supporting the analysis and will help the reader better understand how the modeling objectives stated above are realized. Based on this presentation, the discussion will then focus on how the model captures the various scenarios composing the mission.

The following sections observe and analyze the quality of the automatically generated analysis model. First we will investigate the accuracy of the model in capturing the functional relationships and the subsystem composition. This section is concluded by a critical analysis of the automated model. This analysis support the critical assessment of some of the hypothesis set forward in this thesis.

6.5.1 **Overview of the Analysis Model and Introduction of Default Components**

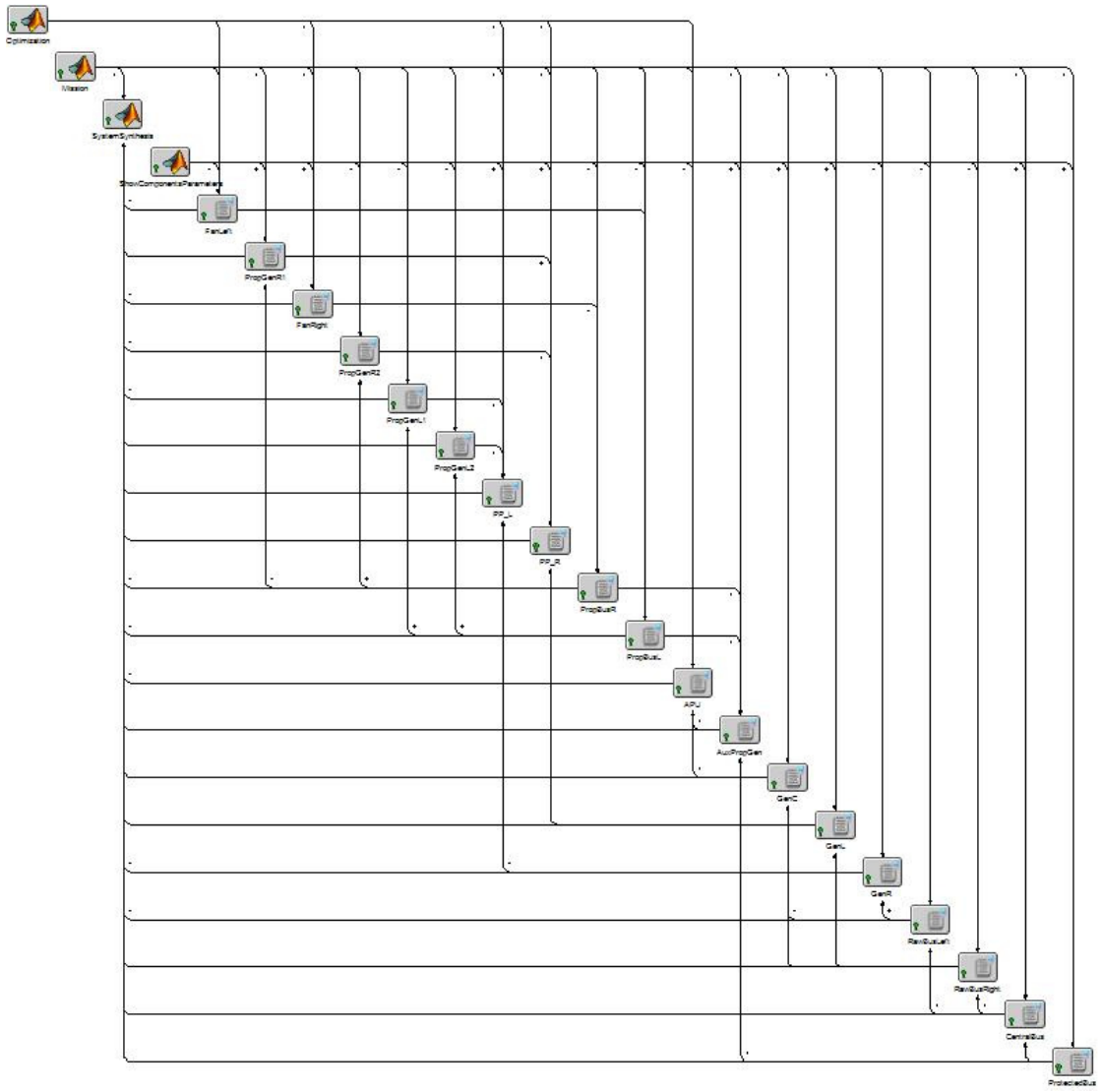


Figure 222: Analysis model composition overview

The resulting architecture analysis model was composed of 23 model components. The top four models (with the red Matlab symbol in Figure 222) were components which are necessary regardless of the architecture concept. The other 19 elements (with the script symbol) represent the modeling bricks which were automatically imported from the Analysis Server library and integrated by the Builder.

It is important to note that the model represented above is the implementation of the analysis optimization problem, stated in equation (53) reproduced below. The 19 subsystem sizing models represented in grey in Figure 222 are the embodiment of the third constraint (*SubsysSizing*()). All the links between the boxes represent the *Structure*() constraints.

$$\underset{\Gamma_{\bullet}}{Max} \quad OpRange\left(W_{\bullet}, \overline{FB}_{\bullet}\right) \tag{53}$$

Subject to:

- $\left[\overline{R}_{\bullet}, \overline{Spe}\right] = Structure\left(\overline{R}_{mission}, \overline{Spe}_{mission}, X_{arch}, \overline{IndFctReq}_{\bullet}\right)$
- $\left[W_n, \overline{FB}_n, \overline{IndFctReq}_n\right] = SubsysSizing\left(\Gamma_n, \overline{R}_n, \overline{Spe}_n\right)$ for $n \in [1, N]$

This section is going to focus on the first four model elements which provide the basis necessary to construct the architecture model. The first paragraph presents the *Optimization* block which provides the interface necessary to easily implement the architecture-level optimizer. The second describes the role of the *Mission* block. The third will focus on the *ShowComponents* block which centralizes the control of display functionalities. This section is concluded by the introduction of the *SystemSynthesis* block which estimates the architecture performance based on subsystem attributes.

6.5.1.1 Implementation of the Architecture-Level Optimizer

The first element is the *Optimization* block. This block is used as an interface between the subsystem sizing models and the architecture optimizer. The optimizer (implemented through the built-in gradient-based ModelCenter optimizer) can be directly plugged into the input variables of this block. These input variables are used to receive the optimizer values for the priority variables. The output variables distribute them to subsystems via connections automatically implemented by the Builder. The fundamental structure of this block is architecture generic but some aspects are specific to the architecture composition.

The interfaces of the module are generically defined, as its input and output variables are resized by the builder for each architecture concept in order to identify the appropriate number of priority variables. Some content of the code implementing the optimizer block is not necessarily generic and may need to be revisited to accommodate different compositions. For instance the *Optimization* block implemented for the test case receives two array variables from the optimizer: *OptimizerInputFan* and *OptimizerInputPP*. These arrays receive the priority factor array *gammas*. The composition specific section makes sure that all subsystems sharing the same type are optimized in the same fashion. This formulation is also an architectural choice. Hence the *Optimizer* block will assign the same priority variables for all power plants. The architecting team could also choose to optimize each power plant differently. If we consider the left and right power plant, for instance, it would make no sense using different optimization strategies for their optimization since they are expected to be symmetric. Therefore, adapting the way the priority variables are centralized allows for the simplification of the optimization process at the architecture-level by decreasing the number of optimization variables.

In order to optimize the priority variables, the ModelCenter built-in gradient based optimizer is used. This optimizer was chosen based on observations made in experiment 3.2. In this experiment we observed that the Coordinated Optimization approach smoothes out the design space (by eliminating local optimum generated by the subsystem trade-offs). Therefore, a gradient based optimization approach was deemed appropriate for this design problem.

6.5.1.2 Mission Block

This block was described earlier as part of the "Mission Requirement Definition" section. This block is fully automated and provides a means for the quantification of the boundary function requirements. The builder constructs its input arrays containing a

description of the operational scenarios (imported from the SysML model – more specifically from the activity diagrams). These operational scenarios are then translated into quantified boundary function requirements. Their values are then dispatched to the correct subsystem model to define the requirements constraining its sizing.

In addition to the mission requirements this block was also defining the functional characteristics (voltage, limiting shaft speeds, etc…) associated with each function type. These variables can potentially be optimized directly by the architecture-level optimizer because they are conceptual parameters defining the architecture itself (in other words, they are not subsystem design parameters). This optimization opportunity was not explored in this test case due to time constraints.

6.5.1.3 ShowComponents Block

This block provides a centralized means for controlling the display function included in each subsystem sizing model. This block defines the array show of size N by 3, where N refers to the total number of subsystem models. Each subsystem model receives a trigger variable which launches a plotting function. An example of the *show* array is presented in the following table.

Table 38: Example of the show array

Subsystem name	Subsystem Type	Trigger
FanLeft	ElectricFan	0
PropGenR1	DCG	1
FanRight	ElectricFan	1
PropGenR2	DCG	0
PropGenL1	DCG	0
PropGenL2	DCG	0
PP_L	PowerPlant	1
PP_R	PowerPlant	0
PropBusR	DCBus	0
PropBusL	DCBus	0
APU	PowerPlant	0
AuxPropGen	DCG	0
GenC	VFG	0
GenL	VFG	0
GenR	VFG	0
RAwBusLeft	VFACBus	0
RawBusRight	VFACBus	0
CentralBus	FFACBus	0

The values stored in the first two columns only provide indications with regards to the list of subsystems composing the architecture. The token on the third column corresponds to the trigger variable for the subsystem assigned in the line. This array provides a user interface for specifying which subsystem information should be displayed on the screen (in order for the architecting team to make a decision based on subsystem information). In order to implement this functionality, as the builder is importing new analyses into the environment, it automatically updates this array and connects the correct value from the *Show* array to the trigger in the sizing model.

6.5.1.4 SystemSynthesis Block

The *SystemSynthesis* block defines the architecture-level attributes. In equation, this block is in fact evaluating the objective function $\left(OpRange\left(W_{\bullet}, \overline{FB}_{\bullet}\right)\right)$.This definition is based on the synthesis of subsystem attributes. The list of attributes, necessary to the synthesis, was first identified in the definition of the subsystem type (see Figure 184 reproduced below). For instance, the attributes *W* and *HR* of the subsystem of type *MyMotorType* (not used in this test-case) are declared as necessary to the architecture synthesis. The links necessary to transferring this information are implemented automatically by the Builder.

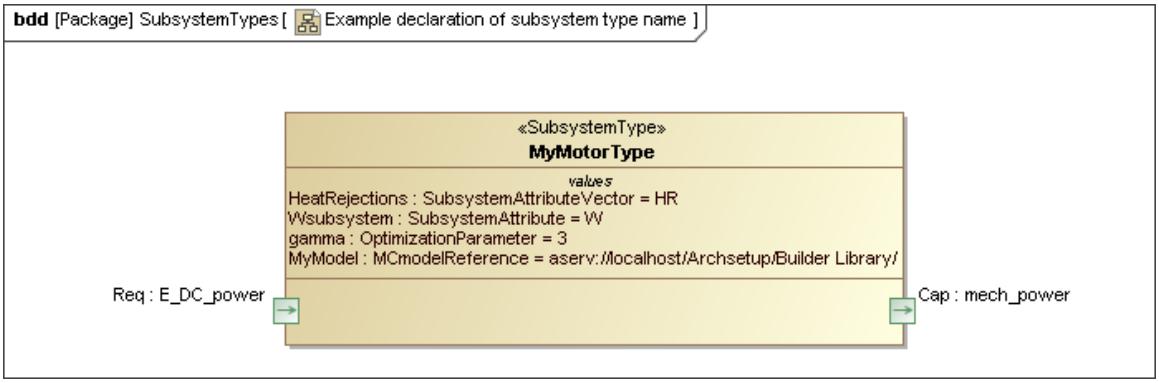


Figure 184: Example designation of the sizing model in the AS library (reproduced)

In the Test-Case the objective of the design problem is to maximize the operational range of the aircraft. In order to do so, it was necessary to estimate the operational range based on subsystem attributes. The range was defined by assuming a fixed total weight for the aircraft. Since the weights of subsystems are free to fluctuate, the mass of fuel on board is going to be adjusted to meet the fixed total weight. In other words every additional kilogram for the subsystems will decrease the amount of fuel that can be carried onboard by one kilogram. This type of objective was chosen because it provides an interesting trade-off between subsystem weight and efficiency. Heavier subsystems will limit the range because less fuel can be carried on board, but less efficient subsystems will also limit the range because they imply a higher fuel burn rate.

The other components composing the architecture model (shown in grey in Figure 222) were imported automatically from the Analysis Server Library by the architecture model builder. These models are the subsystem sizing models. Each of them represents a different subsystem in the architecture. The web of lines between the models represents the variable connections. Overall 3381 connections were automatically created in order to implement the architecture analysis model.

6.5.2 **Representation of the Mission**

Several reasons can explain the number of the connections created to implement this model. The first reason is the representation of the mission. In this model, 20 flight sequences were represented. Each of these sequences corresponds to a possible path between the "start" and "end" nodes in Figure 213 (reproduced below).

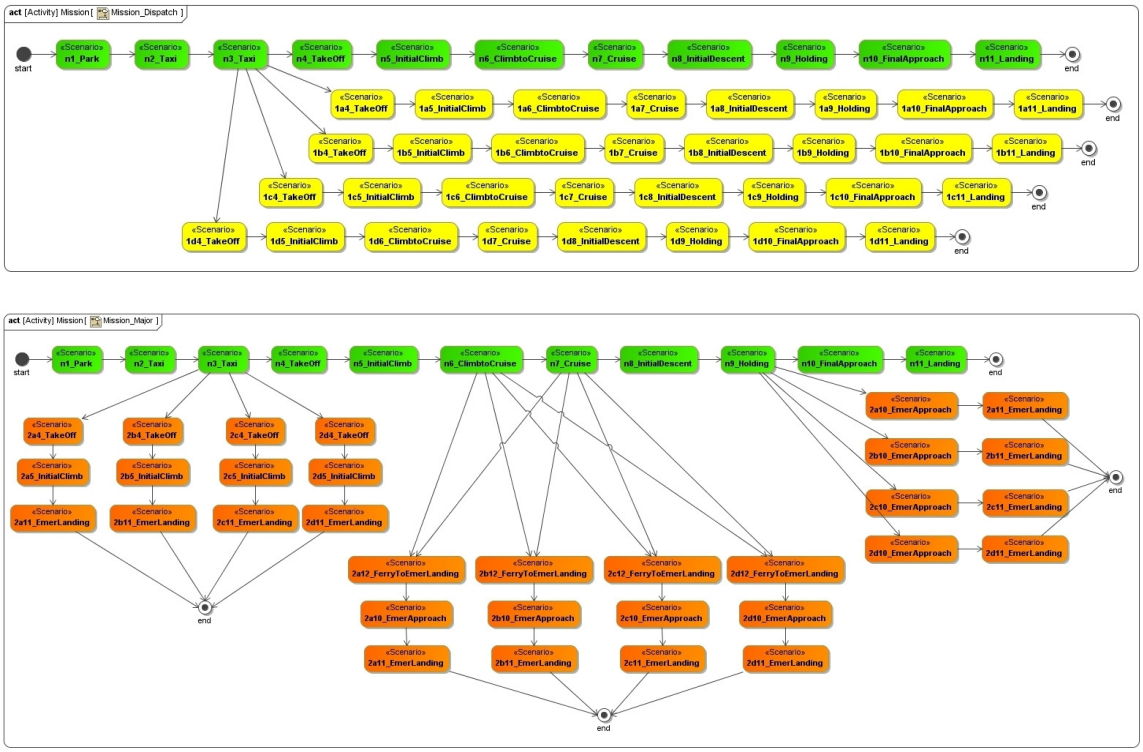


Figure 213: Mission sequences - Architecture 1 (reproduced)

The sequences were identified by the builder which explored the tree structure implied by these activity diagrams. These sequences are represented in the automated report produced by the Builder, shown in Figure 223.

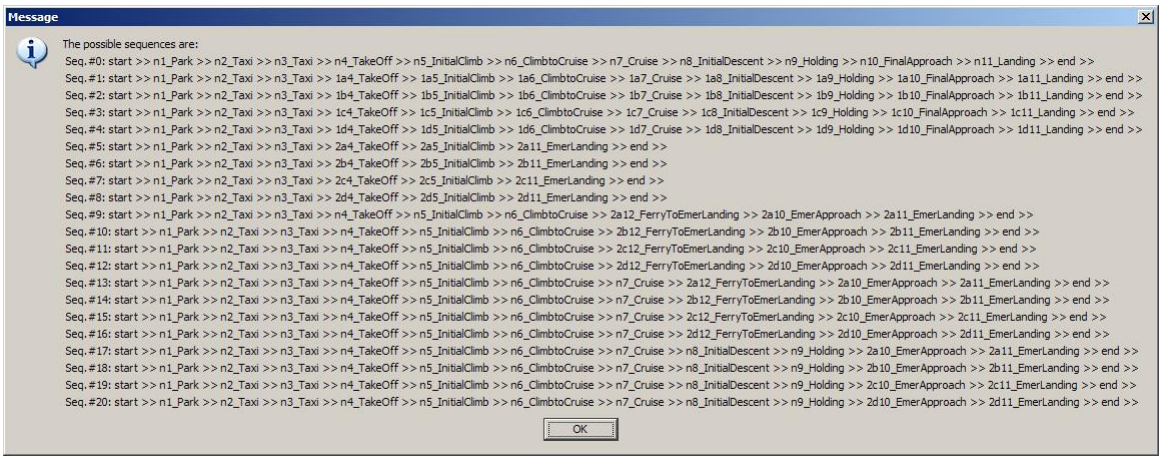


Figure 223: Representation of all possible mission sequences

As prescribed by the methodology, the mission states are defined by lining up all possible sequences. Based on this approach, 199 operational states were defined. The operational variables (state based attributes – e.g. fuel flow- , functional requirements – e.g. power requirements - and specifications – e.g. Nominal voltage) were connected for the representation of each state. In addition to the operational states, the total number of states also includes blank states used to separate the operational states associated with different sequences. Therefore, the operational variables are defined by arrays of size 219 (199 operational states plus 20 blank states).

For each operational state, the connections necessary to represent the architecture configuration associated with the scenarios were implemented. In the architecture concept presented earlier, each configuration includes about 20 functional relationships. The functional flows implied by these functions imply between three and five connections. Therefore, the number of connections necessary to the implementation of an analysis model capable to represent the performance under so many operating scenarios is massive.

6.5.3 **Implementation of Functional Relationships**

Amongst the 3381 connections in the model, over 2708 were implemented to represent functional relationships. Together these connections implement the constraints specified by the *Structure*() function in the overall analysis optimization problem. The automated implementation of these constaints constitutes one of the most important abilities provided by the proposed methodology. In this section, we shall observe a subset of these connections to observe the accuracy of the Builder in capturing these constraints.

First, we shall observe the destination of the requirements emitted by the *ProtectedBus*. This subsystem is connected to the *AuxiliaryPropGen* (generator located on the APU) for ground take-off, approach and landing operations. In airborne operations (other than approach) this bus is connected to the *CentralBus*. In the first configurations the power received is direct current while in the other configuration the power received is fixed frequency current. The following figure shows the connections of the *ProtectedBus* sizing model.

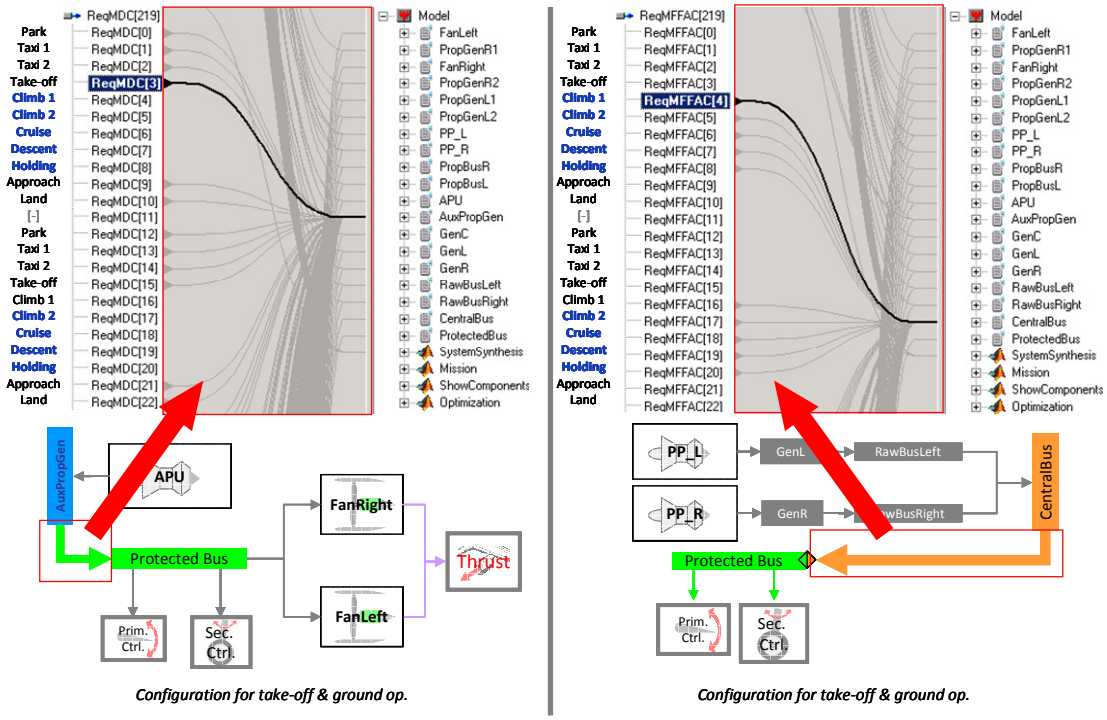


Figure 224: Implementation of a functional relationship

On the top of the figure, we can see the connections in MC for the variables *ReqMDC* (which quantifies the amount of DC power required from *ProtectedBus*) and *ReqMFFAC* (which quantifies the power requirements for FFAC power for *ProtectedBus*). We can see that the information is emitted from *ReqMDC* to *AuxPropGen* during park, taxi, take-off, approach and landing operations. In climb, cruise and descent the information the *ReqMDC* is disconnected (hence no requirements are defined) and the *ReqMFFAC* variable is active and connected to *CentralBus*. This switch in information shows that the builder has been able to capture the variations in the functional relationship of the *ProtectedBus* accurately.

Based on these relationships, the bus *ProtectedBus* can be sized using the sizing relationships provided in appendix E. Using the functional requirements from the ensemble of the operational scenarios, in which the subsystem is susceptible to operate, facilitates this sizing process. This observation is true for all subsystems composing the architecture. Using their functional relationships in all operating scenarios allows for the characterization of the ensemble of the requirements constituting the mission of the subsystem. Therefore, it is based on a complete description of the subsystem mission that the sizing model can base its analysis. If this description turns out not to be complete then there are two possible explanations. First, the lack of completeness may result from an inappropriate definition of functional flows (in the "Preparation of Subsystem Knowledge" activity). Secondly, an operational scenario was not identified which would require revisiting the "Architecture Concept Definition" (i.e. the SysML model).

It is important to note that this observation provides strong evidence supporting the fact that *Research Question 2.2.2* (*How do we translate the architecture structure into a sizing MDA I/O structure?*) is addressed by *Hypothesis 2.1* (*The functional relationships between two subsystems characterize the flow of information between their sizing processes*).

6.5.4 **Observations on the Model Composition**

Based on the observation above we can conclude that the Builder is able to produce a model which allows for a functionally accurate analysis of the architecture concept. We shall now consider the composition of the model.

As previously mentioned the model is composed of 23 model components. Besides the four baseline model components (*Optimizer*, *Mission*, *SystemSynthesis* and *ShowComponentParameters*) the remaining 19 model components correspond to the modeling bricks representing the subsystem sizing models. The connections between the model components were described in previous sections. The fact that all subsystem attributes necessary to the system synthesis could be defined support the fact that *Research Question 2.2.1* (*How do we translate the architecture composition into a sizing MDA composition?*) can be properly addressed by *Hypothesis 2.2* (*The composition of the analysis model corresponds to the physical composition of the architecture*).

But if we look at the tasks performed by the subsystem sizing models, we can see that the operations performed by some modeling bricks are redundant. This redundancy can be observed for all symmetric subsystems composing the architecture. For instance, the snapshot below presents a close view on eight of the modeling bricks composing the analysis model. These eight bricks are composed of three types of models:

- The sizing model for the electric ducted fan (highlighted in blue)
- The sizing model for the DC generator (highlighted in yellow)
- The sizing model for the power plant (highlighted in green)

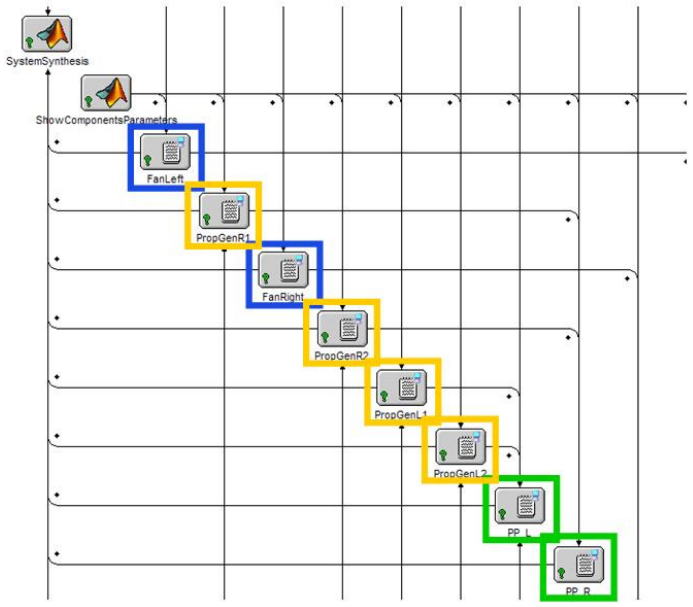


Figure 225: Close up view on model

If we take a closer look at both electric fan designs, we can observe that they are identical. If we now take a step back and observe their role in the architecture concept, we can see that they are identical (they perform the same functions in all scenarios). Therefore, integrating the same modeling brick twice because the subsystem is present twice in the architecture is not an efficient modeling choice.

Similarly the DC generator designs produced by the sizing models are identical. But unlike the electric fans their functions are not identical. In the architecture concept definition phase, we have defined failure scenarios where different combinations of DC generator failures were considered. But given the symmetry requirement, the definition of failure scenarios offset the differences in requirements. Therefore, the requirements imposed on these four DC generators were implemented in a different order, but the overall set of requirements was identical for these four subsystems. The same observation is valid for the power plants.

Based on this observation one can be tempted to say that the most efficient composition of the analysis model can be defined by the list of subsystem types included in the architecture. In order to avoid this misconception, it is important to consider for

instance the fact that the *AuxProGen* is also a DC generator which has a design very different from the one of the four propulsive generators presented earlier. Therefore, the analysis model composition can not be described by the sum of all subsystem types integrated in the architecture concept.

This multiplicity in the subsystem sizing can be traced back to subsystem symmetry. If subsystems are symmetric, the sizing of several subsystems can be grouped into one similar sizing process. Based on this observation, it can be concluded that althoug the approach provided by *Hypothesis 2.2* (*The composition of the analysis model corresponds to the physical composition of the architecture*) addresses *Research Question 2.2.1*, its solution is not the most efficient and should be used as a motivation for future developments taking advantage of subsystem symmetry.

But regardless of this limitation, the ability of builder to translate the conceptual model into an accurate analysis model allows us to conclude that *Research Question 2.1*(*Which aspect of the architecture must be captured to define its analysis model?*) is addressed by *Hypothesis 2*(*The functional, physical and operational description of the architecture provides an unambiguous description of the architecture and facilitates the establishment of the analysis model*).

6.5.5 **Assessment of the Practicality of the Analysis Model**

Based on the observation made earlier, the analysis model produced by the Builder is able to capture accurately many architecture concepts. Therefore, we can conclude that the methodology allows for flexible modeling. We can also say that the methodology is practical from an analysis point of view, because:

- The model is built automatically
- The environment (in this context Model Center) can converge the model to a solution automatically
- The optimization problem is automatically set up and capitalizes on the built in abilities of the modeling environment (Model Center)

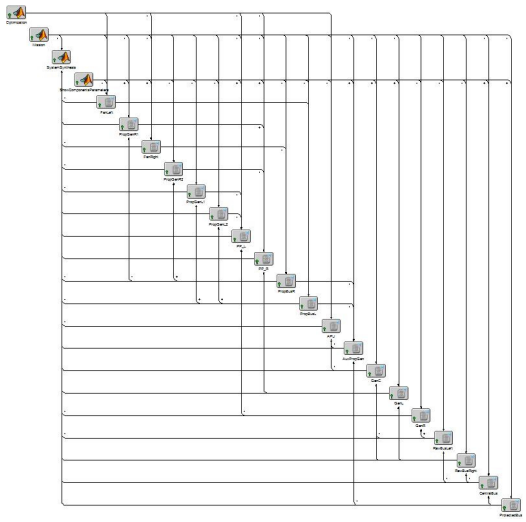


Figure 222: Analysis model composition overview (reproduced)

On the other hand, the practicality in terms of visualizing the results could be improved. It is difficult to use the modeling environment to understand what is going inside the analysis. Unlike SysML which is visual and intuitive, the environment provided by Model Center imposes some limitations in terms of the accessibility of the information. If we considered the overview of the model in Figure 222 (reproduced above), it is not obvious how subsystems mutual requirements constrain their sizing.

Indeed, in order to follow how the requirements propagate from one subsystem to another, a lot of clicking around is necessary.

This visual limitation is by no means a showstopper as is demonstrated by the numerous users of Model Center. Improved visualization and accessibility would greatly facilitate the adoption of such a framework and would greatly facilitate the investigation of the results provided by the analysis.

6.5.6 **Observation on the Architecture Analysis Model Setup Time**

The model used to represent the baseline was based on 23 model components integrated in a web of 3381 connections. These components and connections are necessary to the representation concept and its analysis. Since this model is specific to the architecture concept, this model needs to be redefined for each new concept considered.

Let us assume that on average one minute is necessary to create a connection and 10 additional minutes will be necessary to debug every 100 connections (very optimistic estimation). Based on these estimations, the implementation of the model presented earlier would necessitate 62 hours (3719 minutes). In the previous section it was observed that the time associated with setup of the conceptual model (SysML model) took approximately 4 hours. In order to produce the architecture analysis model, the Builder necessitates and additional 15 minutes. Therefore, we can observe that the methodology allowed reducing the time necessary to the implementation of the analysis from 3719 to 255 minutes (acceleration by a factor of 14). This observation clearly supports the fact that the methodology proposed in this thesis supports the following objective:

Objective 3: Accelerate turn around in architecture concept analysis.

6.6 Architecture Analysis and organization of a Trade-Off

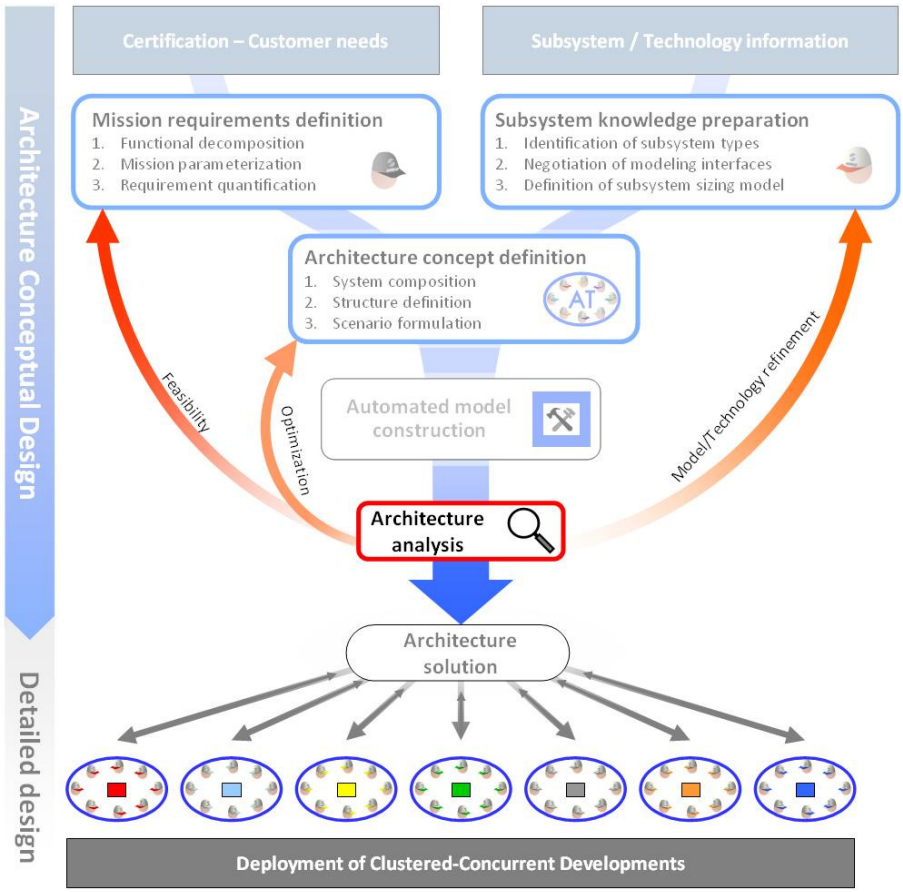


Figure 226: Process overview - Architecture Analysis

This section describes how the analysis model can be used to make architecting decisions. The possible applications of the architecture analysis are certainly not limited to those described in this paragraph. But the objective of the present discussion is to classify the general analysis activities and how they fit in the general methodology proposed in this dissertation. In a very general fashion, one can classify the analysis activities under four general types:

- Model or technology refinement
- Feasibility assessment
- Concept optimization
- Preparation of architecture developments

This present section attempts to describe and comment on these types. For some of them, an example extracted from the test case will be used to illustrate the different forms of analysis. Some of these examples provide quantitative numbers extracted from the model used in the test-case. It is important to keep in mind that the numbers provided in these examples are only shown for the sake of illustration. Therefore the reader is kindly requested to consider these numbers and technical conclusions for their illustrative purposes only, and not as a technical assessment of the distributed turbo-electric aircraft technology, architecture concepts or the mission requirements formulated in Appendix A

6.6.1 **Modeling Refinement Loop**

Several situations will lead to the necessity of refining subsystem models. Some of these situations are that:

- An important interaction was not included in the analysis
- A subsystem model was not valid
- Further investigation is necessary to improve the definition of a model element.

6.6.1.1 Adding a Missing Interaction

The constraints imposed on aircraft subsystems can take different forms. Often in conceptual design, it is impossible to capture all interactions at once. It is therefore necessary to down-select the effects that must be captured in order to have a useful model. But missing a constraining effect will produce misestimated subsystem sizing attributes. It is therefore the role of the experts to monitor that, as the subsystems are sized, nothing indicates that a constraint may be imposed to other subsystem via relationships previously ignored. Similarly, the expert has to verify that the design defined by his sizing model does not need information which is not captured in current functional flows.

The method proposed in this thesis organizes interactions around functionalities. If a missing interaction is detected, it is necessary to determine whether this deficiency is due to:

- An inappropriate definition of a functional flow
- The non-consideration of a functional relationship between subsystems

6.6.1.1.1 Modification of a Functional Flow

In the case where a functional flow was inappropriately defined, the architecting team has to redefine a new functional flow. This task requires revisiting the definition of functional flows. Technically, the functional flows are redefined using the internal block diagram in the subsystem model library. It is important note that as a functional flow is modified, the variable set necessary to characterize the function will be changed. Consequently, the modeling bricks representing subsystems requiring or providing the function will need to be adapted to produce or receive the additional variable. Therefore, having to modify a functional flow requires some degree of negotiation among the architecting team adapting the functional flow and updating the relevant subsystem sizing models (i.e. modeling bricks).

Once the functional flow diagram and modeling bricks are updated, the architecture builder can be used to re-translate the conceptual model. (Note: in this situation, no changes are necessary in the SysML conceptual model representing the architecture concept). The updated functional flow and modeling bricks will be used in the construction of the new analysis model. The process implied by this correction is presented in the following figure.

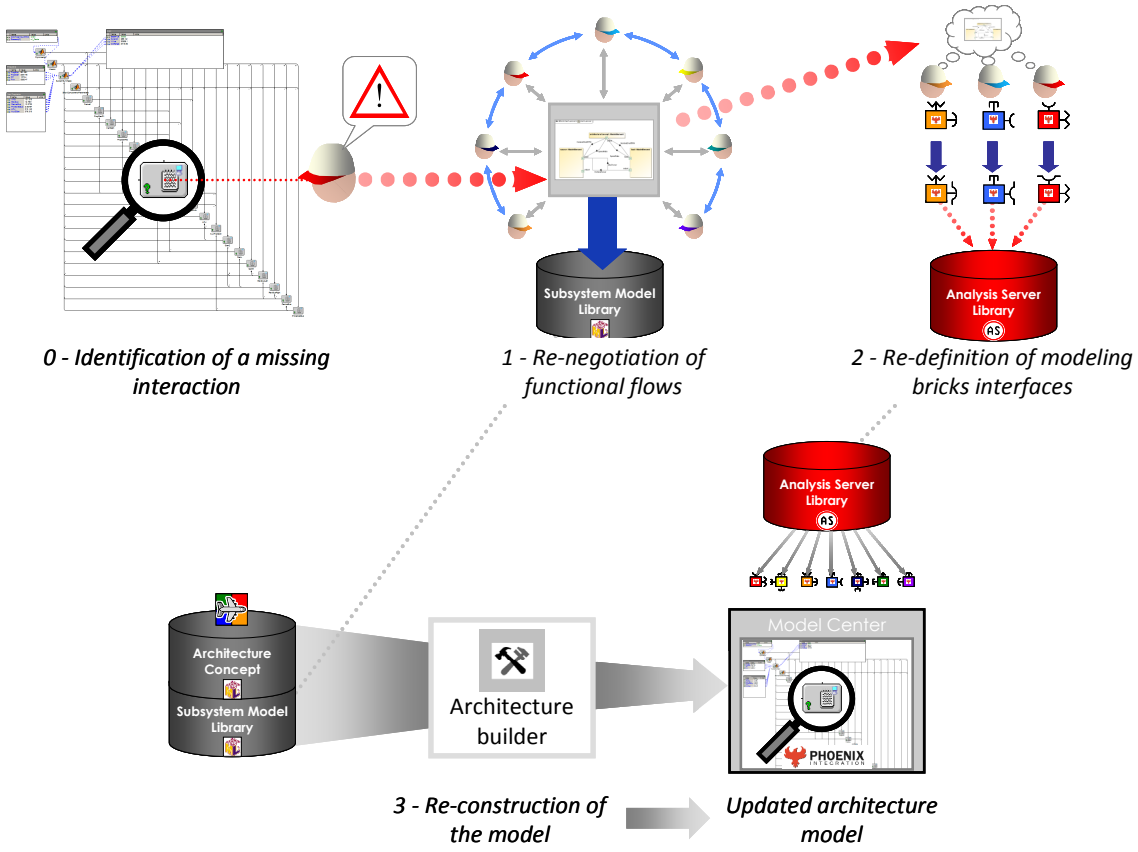


Figure 227: Redefinition of functional flows

6.6.1.1.2 Integration of a new Functional Relationship

In some situations, the missing relationship may result from the fact that a collateral function was not considered. If this function was not defined previously, it is necessary to declare it and specify its functional flow. Once the function is identified, it is necessary to update the functional interfaces of the subsystem types involved in the missing relationship. This is done by adding the function as a capability or requirement port on the functional analysis diagram of the subsystem type (process described in Figure 181). As a consequence of this redefinition, the variable interfaces for the models representing these subsystem types must be changed. The sizing models representing them must be adapted in order to produce or receive the variables associated with this new functional flow assignment.

Once this addition is implemented, it is necessary to update the architecture concept model by adding the missing functional relationships in the relevant configuration diagrams. Based on the updated architecture concept, the Builder can be re-activated and the new analysis model produced. An overview of the process for the integration of new functional relationships is provided in Figure 228.

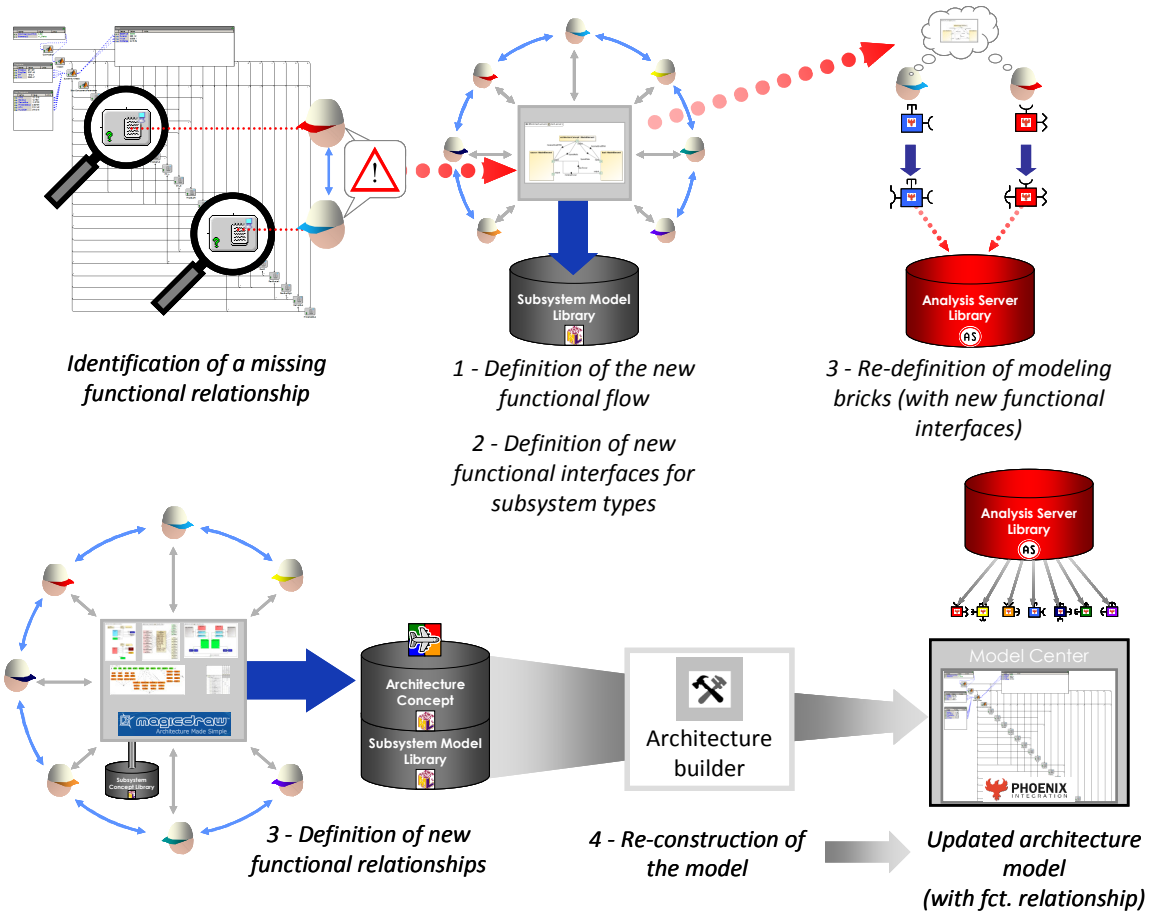


Figure 228: Integration of a new functional relationship

6.6.1.2 Modification or Specialization of Sizing Models

The modification of a subsystem sizing model is necessary for situations where:

- The constraints can not be satisfied within the validity range of the original model
- More accuracy is necessary from the model.

The complexity associated with the modification of a subsystem model depends primarily on the need to adapt its interfaces. If no modifications are necessary this can be

done primarily at the initiative of the subsystem expert. In this circumstance, the expert only needs to update the Matlab function containing the sizing model. The reference of the modeling brick does not have to change and the same architecture analysis model can be used to run the updated subsystem model.

In some situations, the improvement of the model requires more information about the operating environment of the subsystem. This modification requires adapting the functional flows around the newly redefined subsystem model. In that situation, the modification of the subsystem model requires the architecting team to either re-negotiate the definition of function flows or to create new flows. In this situation the redefinition process would become similar to the one presented in Figure 228.

6.6.2 **Feasibility Assessment Loop**

An important aspect of conceptual design is to observe which requirements are technically feasible or which are not. The conceptual design activity will necessarily imply some a priori formulation of needs and preferences which are then formulated as functional requirements and figures of merit for our architecture. In a context where new technology is considered, the understanding of what is feasible, or not, may be difficult to grasp intuitively. As a result, it may be necessary to go back and forth between what was initially expected and what is actually feasible (or acceptable). The analysis model allows for observing the effects of requirements. If a requirement imposes a highly penalizing constraint on a subsystem, the analysis allows for quantifying the penalty implied by the requirement, hence enabling informed decision-making.

This requirement assessment loop can take various forms. In order to illustrate the trade-off opportunities offered by the analysis, we shall consider two typical studies that can be performed with regards to requirement and feasibility assessment. The first that we will consider can be described as a sensitivity analysis of the architecture performance

with regards to mission requirements. The second highlights the potential of the analysis to support requirement negotiations between the experts and the architect.

6.6.2.1 Requirement Sensitivity Analysis (or Detecting the Constraining Requirements)

In Appendix A, eight boundary functions were defined. The parameterization of the mission in phase and depth of criticality required defining 33 requirements for each boundary function. Overall 264 requirements were defined. But they do not all influence subsystems sizing (and by extension architecture/aircraft performance). In order to detect which requirements actively constraints the sizing of the architecture, the analysis can be used to run a simple design of experiments. This design of experiment perturbs each value assigned to the requirements. For each experiment the figure of merit at the architecture level will be observed. If it has changed, it means that the requirement just perturbed is active.

When a requirement is actively driving the sizing of a subsystem, it contributes to the formulation of a sizing scenario. For instance, in the architecture considered in the test case, if the maximum taxi thrust requirement is modified, the APU size will change. Therefore, the taxi thrust is a **constraining requirement**. On the other hand, the control power requirement in take-off (which is also energized by the APU) will not have any influence on the APU size since it was already constrained by another requirement. Therefore in this context the control requirement can be considered as an **opportunistic requirement**. It is qualified as “opportunistic” because it is performed for “free” (i.e. it is using a capability which is already made available for another requirement).

6.6.2.2 Negotiation of Functional Requirements in Conceptual Design

In the conceptual design phase of power architectures, it is fairly common for the subsystem experts to negotiate mission requirements with the aircraft architect. The ability to quantify the impact of requirements enables informed negotiations between them.

To illustrate this type of negotiation let us consider the following situation extracted from the analysis of the baseline architecture. In this architecture, the APU powers the electric fan during taxi operations rather than using the main power plant. This was done in order to decrease the airport noise (by using a smaller turbine), improve fuel consumption and life cycle cost by decreasing the operating time of main power plants. As a result, the size of the APU is primarily constrained by the taxi thrust requirement. Initially the thrust requirement was set at 100kN by the architect. This thrust requirement implied using an APU which would weight about 1200 kg (including the electric generator). Decreasing the taxi thrust requirement to 50kN allows cutting the APU size to 650 kg (including the electric generator). This weight decrease, of over half a ton, improves the operational range by 200 nm. Based on the observation above, the aircraft architect can make informed decisions with regards to the taxi thrust constraints: "Is it worth maintaining high thrust requirements on the ground at the expense of 200 nm of operational range?"

Using this ability to quantitatively support the trade-offs between requirements and subsystem sizing improves the quality of mission requirements by understanding quantitatively their impact on the architecture.

6.6.3 **Optimization loop**

Earlier in this chapter, the design problem underlying this proof of concept was presented as the following optimization problem.

$$\underset{\Gamma_\bullet, X_{arch}}{Max} \quad OpRange\left(W_\bullet, \overline{FB}_\bullet\right) \tag{54}$$

Subject to:

- $\left[\overline{R}_\bullet, \overline{Spe}\right] = Structure\left(\overline{R}_{mission}, \overline{Spe}_{mission}, X_{arch}, \overline{IndFctReq}_\bullet\right)$
- $\left[W_n, \overline{FB}_n, \overline{IndFctReq}_n\right] = SubsysSizing\left(\Gamma_n, \overline{R}_n, \overline{Spe}_n\right)$ for $n \in [1, N]$

This subsection will describe how this overall optimization process is implemented using the model-based architecting process. This optimization takes two forms. The first form of optimization occurs for a fixed architecture concept (X_{arch}) and concerns the trade-offs within a fixed architecture concept ($\Gamma_\bullet$). This level of optimization, referred to as the "subsystem sizing optimization", is automatically implemented by the coordinated optimization method and performed by the analysis model. This level of optimization will be discussed in the first subsection. The second type of optimization concerns the architecture concept optimization and implies different architecture concepts (one engine versus two engines, etc…). Its objective is to identify the one yielding best aircraft performance. This type of optimization, considered within the scope of the architecting team, is considered in the second subsection.

6.6.3.1 Subsystem Sizing Optimization

This optimization loop is automatically performed by the optimization setup included in the architecture analysis model. This setup is based on the Coordinated Optimization method proposed in this thesis. The method which was presented at length in previous chapters will not be discussed again in this section. However the architectural advantages associated with this aspect of the methodology must be highlighted. Therefore

a brief discussion on the application of the Coordinated Optimization on the baseline architecture is provided.

Using the coordinated optimization method, the architecture performance could be drastically improved. With an initial operation range of 4100 nm (using evenly distributed optimization priorities), the optimization of the priority variables enabled improving the performance up to 4864 nm. This improvement by over 18% highlights, once more, the great importance of using optimizer-based sizing models. Further, it is important to note that this capability was enabled by Coordinated Optimization. Without optimizing the sizing priority, the subsystem optimization potential would remain untapped and the results of the analysis would be both inaccurate in estimating the true attributes at the subsystem-level and conservative at the architecture-level.

In this application, the architecture-level optimizer optimizes two sets of priority variables (one for the electric fans, the other for the power plants). As mentioned earlier in this chapter, the same set is used for all fans and a different one for all power plants. The electric fan includes four priority variables (weight, geometric size, energy requirement in cruise, and electric power requirement during peak operation). Since the geometric size of the fan was not considered in this model, the priority factors dedicated to it was defaulted to zero. This elimination decreased the number of fan priority factor down to three parameters. Using the unity constraint on the priority variables (their sum must equal to one), optimizing the fan priority variables could be summarized by two degrees of freedom. The power plant priority set includes only two optimization factors (one for the weight priority, the other for its maximum electric power requirements), the optimization of the power plant only implies one degree of freedom (using the unity constraint). Hence, the implementation of the determination of priority variables required the optimization of three independent factors. The optimizer setup used to implement this process is presented in the following figure.

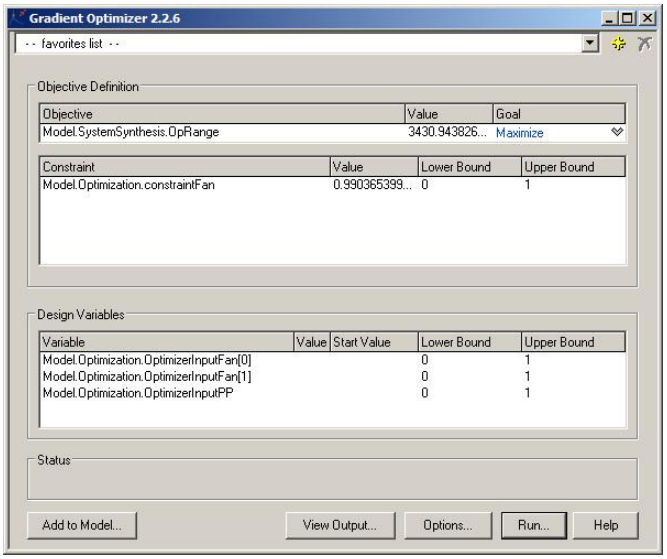


Figure 229: Optimizer setup for coordinated optimization

Note: The optimizer setup presented in Figure 229 is the implementation of the optimization problem shown in equation (53).

In order to illustrate the action of the coordinated optimization, we shall consider the topology of the architecture-level optimizer (similarly to what has been done in experiment 3.2). In order to facilitate the visualization of the 3D optimization topology, one of the degrees of freedom is defaulted. The fan energy priority variable (priority variable dedicated to cruise efficiency) is maintained to its optimal value point. Fixing this variable allows visualizing the topology of the optimizer in two dimensions. The topology represents the weight/power trade-offs of the fans and the power plant.

The topology was defined by running a 20-levels/full-factorial design of experiments to explore the optimizer space. The resulting observations are presented in Figure 230. This figure clearly presents the effects of the internal subsystem trade-offs. It also highlights a discontinuity in architecture performance can result from the trade-off priority of the fan. This discontinuity represents the transition from motor saturated electric fan designs (lighter because the fan is smaller but the motor larger) to fan saturated solutions (more efficient but heavier because the fan size is dramatically increased).

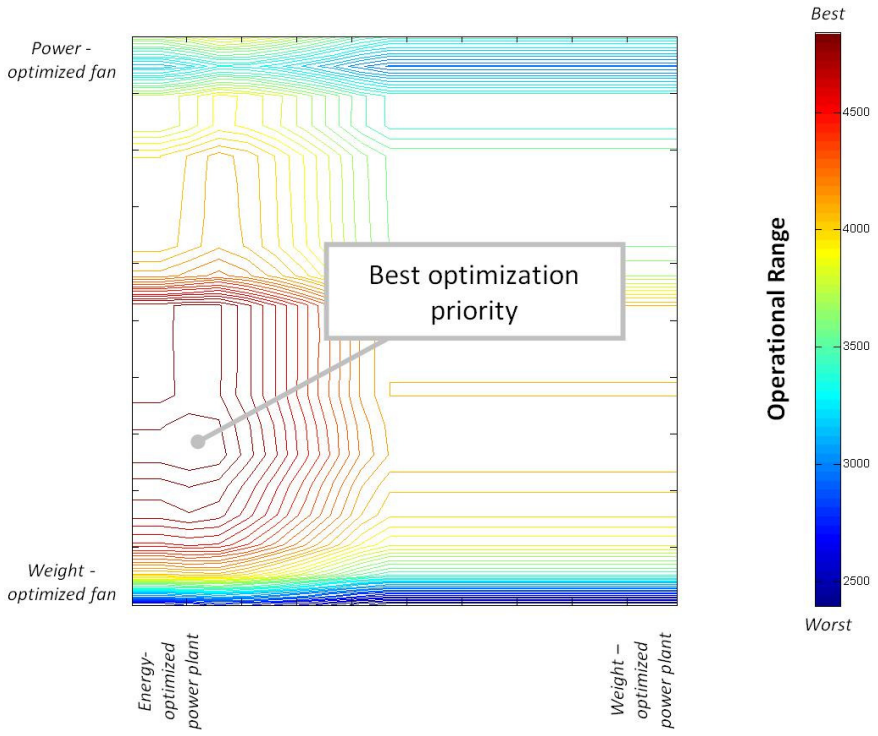


Figure 230: Architecture-level optimizer topology for architecture 1

On the right hand side of Figure 230, the legend presents the range in values for the architecture-level objective. We can see that the worst value was below 2500 nm, while the best was 4864 nm. This observation highlights the essential role of the Coordinated Optimization. If subsystems were sized in an inappropriate fashion, the estimation of overall-architecture performance would be compromised.

6.6.3.2 Architecture Concept Optimization

This aspect of the optimization concerns the search for the best architecture concept for the mission. The primary means to compare architecture concepts is to compare their performance in terms of the system-level objectives. The analysis model synthesizes subsystem attributes to estimate how they perform once integrated. Therefore, when subsystem experts are unanimously confident on the results produced by their models, the system-level figure of merit is the first factor used to evaluate the goodness of the architecture concept. However, in a problem as large as the aircraft power architecture, “not all that counts can be counted”. Although the overall Figure of Merit is

a good starting point to compare architectures in conceptual design, the role of the experts has also a great importance in the assessment of the goodness of the concept. Some architectural aspects are more difficult to quantify in a centralized fashion. Among these aspects, you have things like technical risks associated with a specific subsystem design, difficulty to manufacture the subsystem or operational practicality of a concept. Therefore, it is not uncommon for the quantitative Figure of Merit to take a second place to the negotiation between experts and architects.

The subsystem centric modeling approach proposed in this thesis supports both approaches. It supports quantitative comparisons of architecture concepts by synthesizing subsystem attributes into an overall Figure of Merit. But this synthesis highlights what elements of the architecture penalize it. It does so by providing the subsystem experts with an automated assessment of their subsystem mission and technical solutions. By splitting the analysis at the subsystem-level, it provides the architecting team with a means to observe how subsystems are sized, and how their sizes influence architecture performance. The observation of the subsystem sizing context allows the experts to suggest architecture level improvements.

The modeling approach proposed in this thesis is only using constraints to enforce functional requirements. Therefore, as long as the subsystem sizing models are capable of producing designs that accommodate the requirements, the architecture analysis will be feasible and will return an answer. Thanks to this approach, the analysis will produce subsystem sizing estimations even for bad architecture designs. Hence, even unrealistic requirements and/or poor architectural concepts will be matched with a sized set of subsystems. This ability to converge does not imply that the architecture attributes will be satisfactory. Hence, rather than failing when an unfeasibility is detected, this analysis approach provides evidence documenting the design problem. The results will be able to capture that as formulated the design produces terrible performance (it can have a large

weight figure or an efficiency factor approaching zero, etc…). Hence, as experts review their subsystem attributes, they can identify causes for the architecture underperformance.

Therefore, these results allow the architecting team to trace back the causes for underperformance. By extension, they can also be used to trace back poor performance to the abnormal or unfeasible configurations or requirements causing the problem. These requirements can be the result of either unrealistic aircraft function requirements, or a poorly defined architecture which concentrates too many requirements on the same subsystems (lacking of redundancies).

The architecture concept optimization process, therefore, can be re-presented as an iterative process facilitated by the proposed methodology. This process iterates on the architecture concept definition, automated analysis model construction and architecture analysis. The process is presented in the figure below. This representation highlights the role of the architecting team in the generation of new architecture ideas based on the quantitative analysis provided by the model.

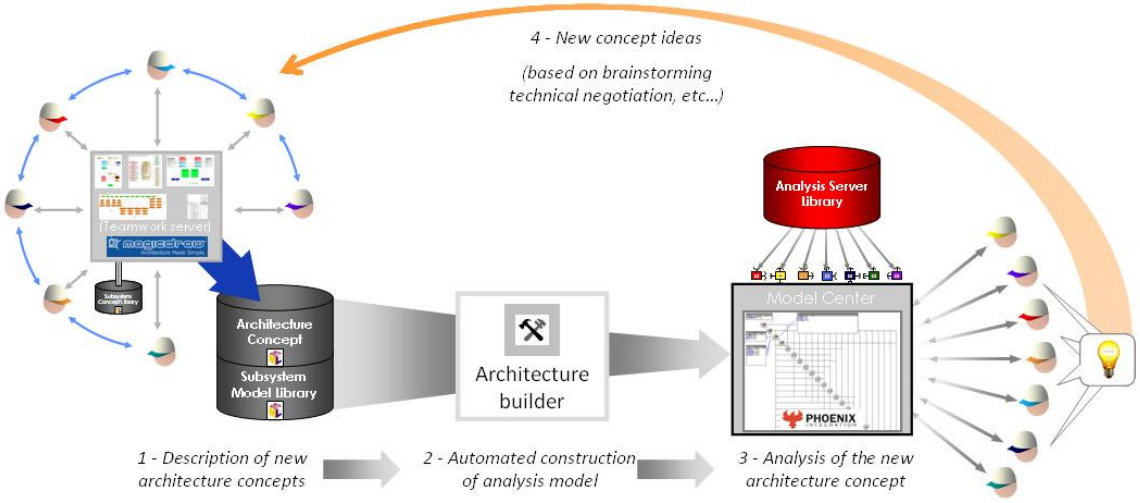


Figure 231: Architecture concept optimization loop

In order to illustrate this approach, we shall consider an example from the turbo-electric architecture. We saw earlier that the architecture is penalized in its performance by the size of the APU. Therefore, a valid architectural idea could be to eliminate the APU from the architecture concept and transfer its functionality to one of the power

plants (step 0 in Figure 231). Based on this idea, a new architecture was implemented in SysML (step 1). This architecture was identical to the baseline architecture with the exception of the APU and its associated VFG which were eliminated. The Builder was used to translate the conceptual SysML model into the analysis model (step 2). This new architecture was analyzed using the new analysis model.

In this example the architecture was redefined as shown in Figure 232 to Figure 236. The first figure presents the composition of the new architecture (note the absence of *APU*). The internal block diagrams following represent the architecture in its take-off and airborne configurations. We can see that in the take-off phases, the *ProtectedBus* is still segregated from the rest of the power architecture at the generator level (it is energized by the *AuxPropGen*). The airborne configuration is strictly similar to the one from architecture 1 (the APU had no role in the airborne configuration in architecture 1).

The failure configurations are defined in a similar spirit to those used for architecture 1. The failure configuration analysis with the electric generators is applied to this new architecture. But in the absence of the *APU*, the *PP_L* takes over the *APU* role in architecture 1. The scenarios and sequences are identical to those of architecture 1.

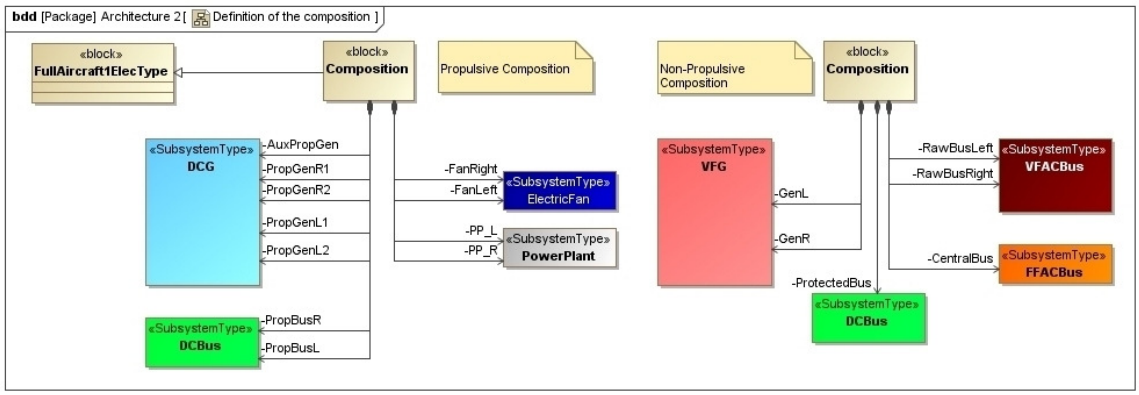


Figure 232: Architecture 2 – Architecture composition

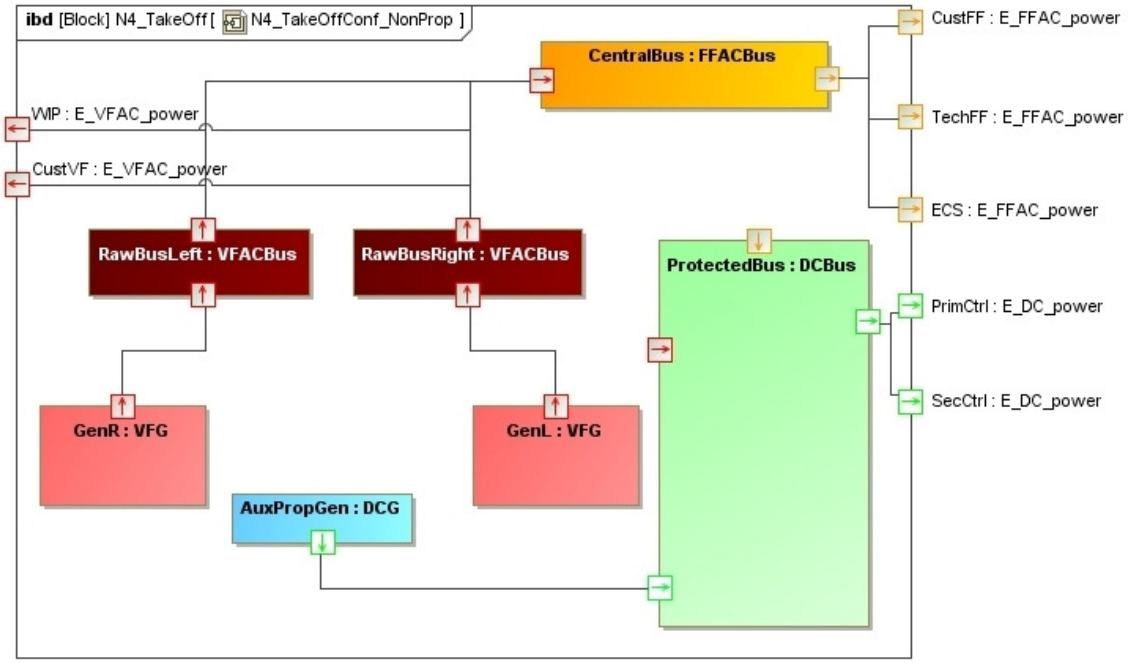


Figure 233: Architecture 2 – Take-off configuration (non-prop. subsystems)

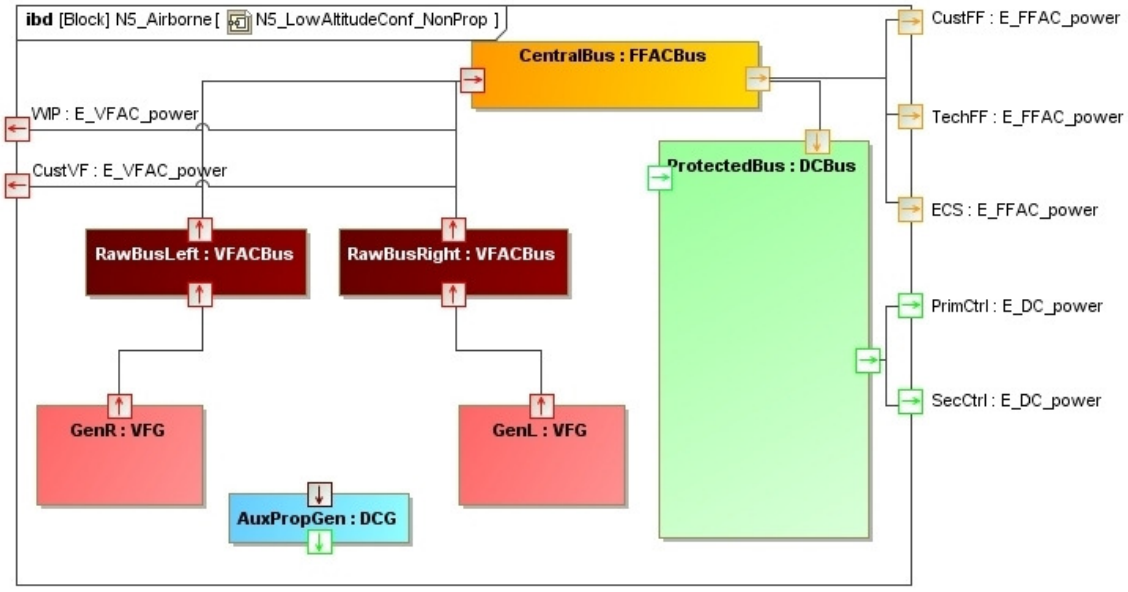


Figure 234: Architecture 2 – Airborne configuration (non-prop. subsystems)

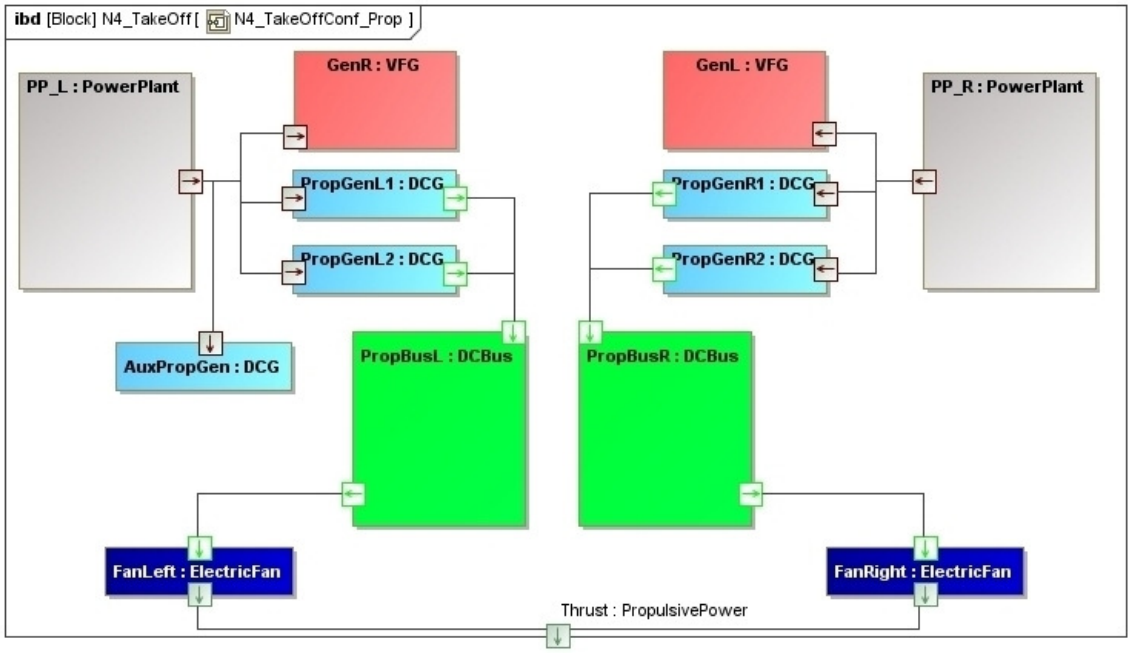


Figure 235: Architecture 2 - Take-off configuration (propulsive subsystems)

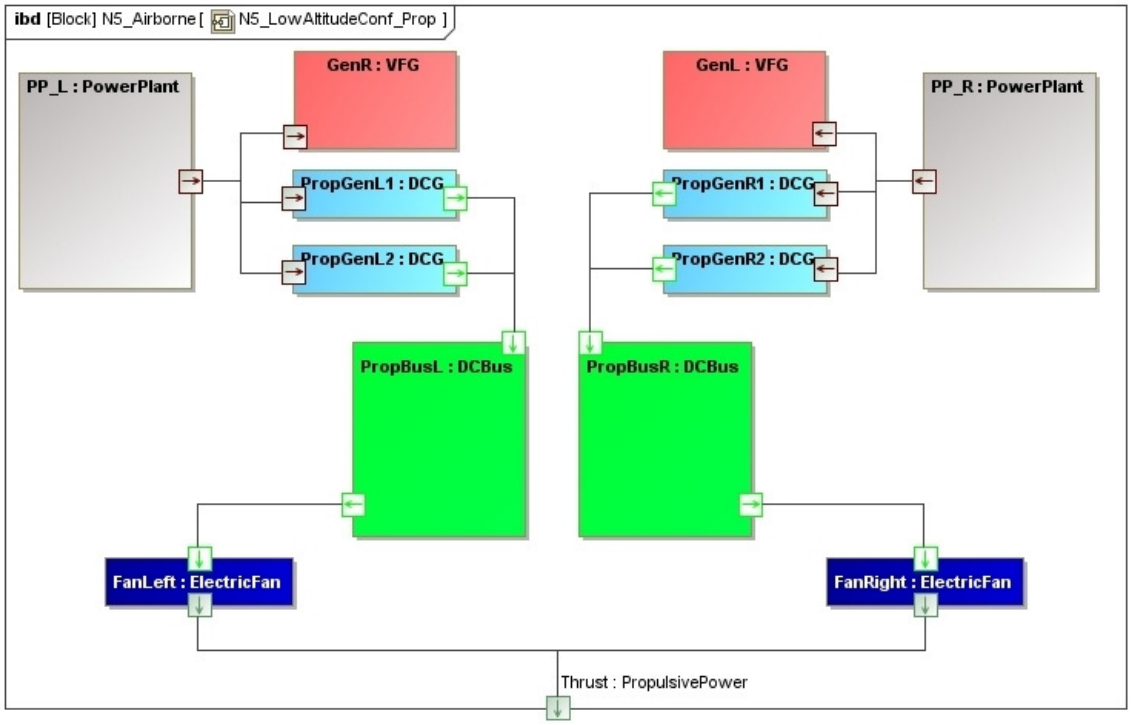


Figure 236: Architecture 2 - Airborne configuration (propulsive subsystems)

The definition of architecture 2 was performed by adjusting the graphs previously defined for architecture 1 (the baseline architecture). The adjustments of the SysML diagrams necessitated about 60 minutes. An additional 15 minutes for the construction of the analysis model was necessary. An overview of the architecture 2 analysis model is presented below.

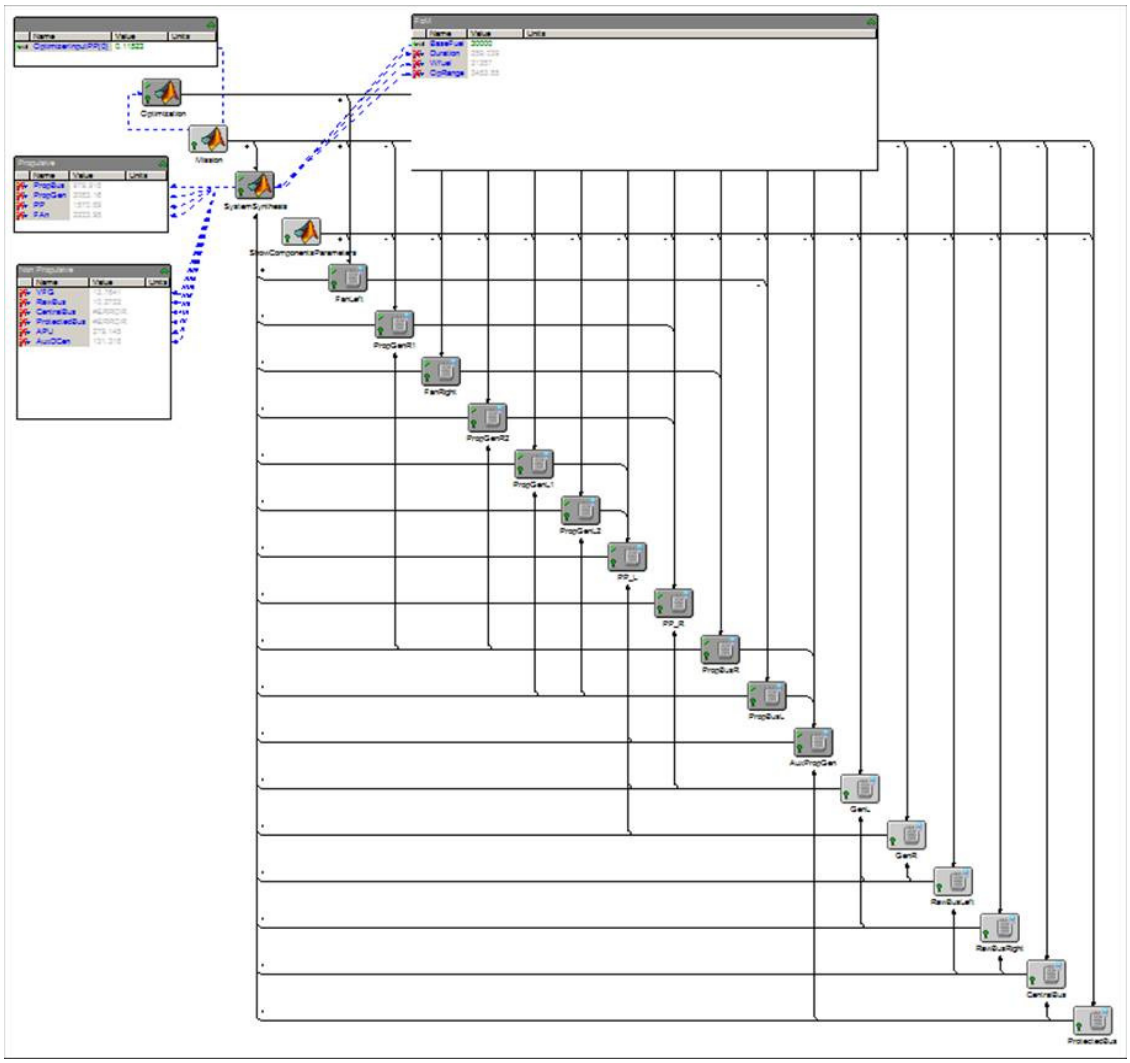


Figure 237: Overview of architecture 2 analysis model

Using the analysis model presented above the operational range associated with architecture 2 could be estimated. This study concluded that the elimination of the APU improved the operation range by 160 nm. This conclusion can now be used to assess the usefulness of integrating an APU. This range benefit can now be traded-off with the

benefits associated with maintaining this subsystem in the architecture. Indeed having an APU offers many advantages not considered by the figure or merit (the operational range) including lower noise for ground operations, lower life-cycle-cost of the (larger) propulsive turbines etc... Never-the-less the quantified assessment provided by the model does allow for better informed decisions toward the optimization of the architecture by the architecting team. This ability directly supports *Research Objective 4*: *Detect architectural opportunities.*

In order to clearly understand, the potential of the model-based architecting methodology, one should consider the time necessary to analyze each architecture concept. Using the methods proposed in this thesis, setting up a new architecture required about 90 min (including the description of the architecture, the construction of the analysis and 15 minutes for making observations on architecture analysis). If the analysis model was to be constructed manually, each architecture concept considered would require weeks.

To further illustrate the acceleration in analysis cycle, let us consider a one month architectural study. Using the model-based architecting methodology, each day about 6 architecture concepts can be analyzed. With five days a week and four weeks in a month, 120 architecture concepts can be explored. In contrast, using a manual approach to architecture analysis would limit the architecture concept exploration to 4 concepts.

Therefore, we can conclude that the model-based architecture methodology provides the architecting team with the ability to explore more architecture concepts necessary to the identification of “better” architectures that exploit technological opportunities.

6.6.4 **Preparation of Architecture Developments**

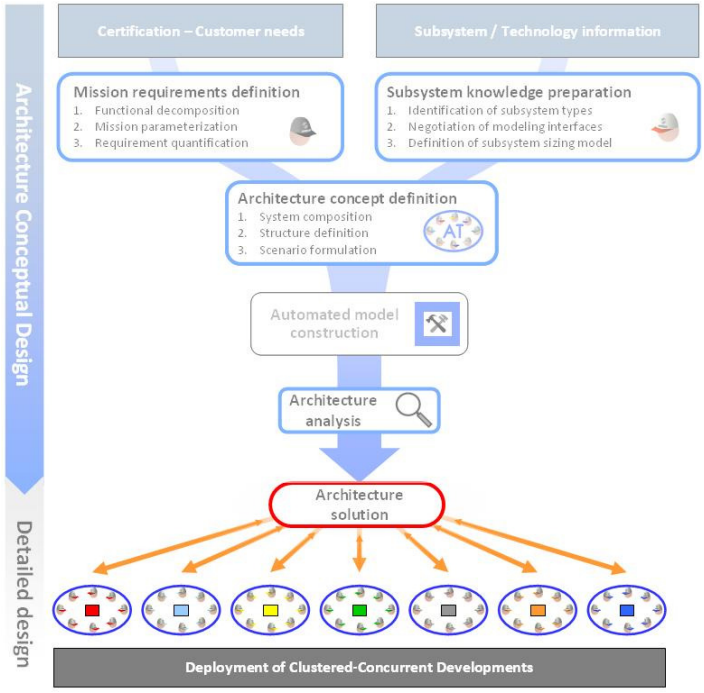


Figure 238: Overview of the architecture solution implementation

The work performed in architecture conceptual design finds its purpose in the preparation of the subsystems developments. As described in the beginning of this thesis, subsystem developments focus on the realization of the subsystems. In the industrial development of a complex system like a commercial transport aircraft, the amount of tasks necessary to perform this development is astronomical. Therefore, in order to archive these developments, a clustered concurrent approach is necessary. The deployment of the clustered concurrent approach necessitates a clear definition of the subsystem design problem.

In the methodology proposed in this thesis, the coordinated optimization method allows for the formulation of clear but flexible subsystem design problems. These design problems allow for the initiation and steering of subsystem developments. In order to illustrate the ability of the methodology to produce the design problem necessary to subsystem developments, an example is provided. This example describes the design

problem for the electric fans in architecture 1. This example is followed by a discussion of the proposed methodology.

6.6.4.1 Formulation of the Subsystem Design Problem – an Example

The definition of each subsystem design problem can be derived from the input variables received by its sizing model. In the example of the electric fan, a list of its input variable is defined in the following table.

Table 39: Input variables to the electric fan sizing model

Variable Name	Description
Dt	Time step
Cruisek	Rank of cruise scenario
show	Switch variable displaying internal information
gamma	Priority factors (weight, diameter, energy, power)
Alt	Flight Altitude
MN	Flight Mach number
ThMax	Max thrust required
Thnom	Average thrust required

As described in previous chapters the design problem takes the form of an optimization problem with an objective function and constraints. The constraints formulate the non negotiable requirements. In the subsystem context the non-negotiable requirements correspond to the functional requirements conditioning the integration of the subsystem. This element is non-negotiable because a subsystem design is acceptable if, and only if, it can perform its function inside the architecture. Therefore, in order to formulate the constraints, it is necessary to consider the variables associated with the functional relationship of the subsystem. In the example of the electric fan, its functional role is to provide thrust. The input variables qualifying this relationship are *Alt, MN, ThMax,* and *ThNom* (as defined by the functional flow). *Alt* and *MN* define the functional specifications, *ThMax* and *ThNom* the functional requirements. Their values are listed in Table 40.

Table 40: Functional Constraints for the electric fan

Scenarioname	DT	Altitude	MH	ThMax	ThNom
n1_Park	60	0	0	0	0
n2_Taxi	15	0	0.04	25000	11500
n3_Taxi	15	0	0.04	25000	11500
n4_TakeOff	5	0	0.04	153500	149000
n5_InitialClimb	10	1500	0.95	106500	106500
n6_ClimbtoCruise	15	27500	0.75	35000	21000
n7_Cruise	90	35000	0.78	34000	23500
n8_InitialDescent	15	10000	0.54	96000	21500
n9_Holding	10	10000	0.45	39000	0
n10_FinalApproach	5	0	0.3	127000	0
n11_Landing					
n1_Park	60	0	0	0	0
n2_Taxi	15	0	0.04	25000	11500
n3_Taxi	15	0	0.04	25000	11500
n4_TakeOff	5	0	0.04	153500	149000
n5_InitialClimb	10	1500	0.95	106500	106500
n6_ClimbtoCruise	15	27500	0.75	35000	21000
n7_Cruise	90	35000	0.78	34000	23500
n8_InitialDescent	15	10000	0.54	96000	21500
n9_Holding	10	10000	0.45	39000	0
n10_FinalApproach	5	0	0.3	127000	0
n11_Landing					
n1_Park	60	0	0	0	0
n2_Taxi	15	0	0.04	25000	11500
n3_Taxi	15	0	0.04	25000	11500
n4_TakeOff	5	0	0.04	153500	149000
n5_InitialClimb	10	1500	0.95	106500	106500
n6_ClimbtoCruise	15	27500	0.75	35000	21000
n7_Cruise	90	35000	0.78	34000	23500
n8_InitialDescent	15	10000	0.54	96000	21500
n9_Holding	10	10000	0.45	39000	0
n10_FinalApproach	5	0	0.3	127000	0
n11_Landing					

Scenarioname	DT	Altitude	MH	ThMax	ThNom
n1_Park	60	0	0	0	0
n2_Taxi	15	0	0.04	25000	11500
n3_Taxi	15	0	0.04	25000	11500
n4_TakeOff	5	0	0.04	74500	53000
n5_InitialClimb	10	1500	0.35	98000	53000
2a1_EmetLanding	5	0	0.3	78000	0
n1_Park	60	0	0	25000	11500
n2_Taxi	15	0	0.04	25000	11500
n3_Taxi	15	0	0.04	74500	53000
2a4_TakeOff	5	0	0.04	74500	53000
2a5_InitialClimb	10	1500	0.35	98000	53000
2a11_EmetLanding	5	0	0.3	78000	0
n1_Park	60	0	0	25000	11500
n2_Taxi	15	0	0.04	25000	11500
n3_Taxi	15	0	0.04	74500	53000
2a4_TakeOff	5	0	0.04	74500	53000
2a5_InitialClimb	10	1500	0.35	98000	53000
2a11_EmetLanding	5	0	0.3	78000	0
n1_Park	60	0	0	25000	11500
n2_Taxi	15	0	0.04	25000	11500
n3_Taxi	15	0	0.04	74500	53000
n4_TakeOff	5	0	0.04	74500	53000
n5_InitialClimb	10	1500	0.35	98000	53000
n6_ClimbtoCruise	15	27500	0.75	35000	21000
2a8_FerryToEmetLanding	20	10000	0.54	30000	21500
2a9_EmetApproach	10	10000	0.45	30000	0
2a11_EmetLanding	5	0	0.3	78000	0
n1_Park	60	0	0	25000	11500
n2_Taxi	15	0	0.04	25000	11500
n3_Taxi	15	0	0.04	74500	53000
n4_TakeOff	5	0	0.04	74500	53000
n5_InitialClimb	10	1500	0.35	98000	53000
n6_ClimbtoCruise	15	27500	0.75	35000	21000
2a8_FerryToEmetLanding	20	10000	0.54	30000	21500
2a9_EmetApproach	10	10000	0.45	30000	0
2a11_EmetLanding	5	0	0.3	78000	0

Scenarioname	DT	Altitude	MH	ThMax	ThNom
n1_Park	60	0	0	0	0
n2_Taxi	15	0	0.04	25000	11500
n3_Taxi	15	0	0.04	25000	11500
n4_TakeOff	5	0	0.04	153500	149000
n5_InitialClimb	10	1500	0.95	106500	106500
n6_ClimbtoCruise	15	27500	0.75	35000	21000
2a8_FerryToEmetLanding	20	10000	0.54	30000	21500
2a9_EmetApproach	10	10000	0.45	30000	0
2a11_EmetLanding	5	0	0.3	78000	0
n1_Park	60	0	0	25000	11500
n2_Taxi	15	0	0.04	25000	11500
n3_Taxi	15	0	0.04	153500	149000
n4_TakeOff	5	0	0.04	153500	149000
n5_InitialClimb	10	1500	0.95	106500	106500
n6_ClimbtoCruise	15	27500	0.75	35000	21000
n7_Cruise	90	35000	0.78	34000	23500
n8_InitialDescent	15	10000	0.54	96000	21500
n9_Holding	10	10000	0.45	39000	0
2a10_EmetApproach	5	0	0.3	127000	0
2a11_EmetLanding					
n1_Park	60	0	0	0	0
n2_Taxi	15	0	0.04	25000	11500
n3_Taxi	15	0	0.04	153500	149000
n4_TakeOff	5	0	0.04	153500	149000
n5_InitialClimb	10	1500	0.95	106500	106500
n6_ClimbtoCruise	15	27500	0.75	35000	21000
n7_Cruise	90	35000	0.78	34000	23500
n8_InitialDescent	15	10000	0.54	96000	21500
n9_Holding	10	10000	0.45	39000	0
2a10_EmetApproach	5	0	0.3	127000	0
2a11_EmetLanding					

Scenarioname	DT	Altitude	MH	ThMax	ThNom
n1_Park	60	0	0	0	0
n2_Taxi	15	0	0.04	25000	11500
n3_Taxi	15	0	0.04	25000	11500
n4_TakeOff	5	0	0.04	153500	149000
n5_InitialClimb	10	1500	0.95	106500	106500
n6_ClimbtoCruise	15	27500	0.75	35000	21000
n7_Cruise	90	35000	0.78	34000	23500
n8_InitialDescent	15	10000	0.54	96000	21500
n9_Holding	10	10000	0.45	39000	0
2a10_EmetApproach	5	0	0.3	127000	0
2a11_EmetLanding					
n1_Park	60	0	0	0	0
n2_Taxi	15	0	0.04	25000	11500
n3_Taxi	15	0	0.04	25000	11500
n4_TakeOff	5	0	0.04	153500	149000
n5_InitialClimb	10	1500	0.95	106500	106500
n6_ClimbtoCruise	15	27500	0.75	35000	21000
n7_Cruise	90	35000	0.78	34000	23500
n8_InitialDescent	15	10000	0.54	96000	21500
n9_Holding	10	10000	0.45	39000	0
2a10_EmetApproach	5	0	0.3	127000	0
2a11_EmetLanding					

A further simplification can be applied to this information by summarizing, it to the maximum requirements associated with each flight condition. These maximum requirements define the critical sizing scenarios, which summarize the most constraining flight conditions.

Table 41: Critical sizing requirements

Altitude	Mach #	Peak Thrust
0	0.04	153500
1500	0.35	106500
10000	0.45	39000
10000	0.54	36000
15000	0.71	42000
27500	0.75	35000
35000	0.78	34000

The functional flow can be used to pre-identify the information that will need to be documented to guide the development process of the subsystem. Forcing the expert to negotiate functional flows for the establishment of modeling interfaces enforces rigor in the formulation of information needs between subsystem domains. This rigor is necessary for modeling purposes, but it is also very useful in preparing the exchange of information essential to subsystem developments. Therefore, this aspect of the methodology supports *Research Objective 5* (*Prepare a framework guiding detailed design developments*).

Beyond the non-negotiable requirements associated with its functionality, the subsystem design problem must specify an objective function. The objective function specifies the optimization priority for the subsystem. As discussed earlier in this dissertation, this objective function attempts to formulate how the subsystem should be optimized in order to contribute to the optimization of the architecture.

The formulation of the objective function can be defined based on the optimal value for the priority variables (defined via the coordinated optimization of the architecture). In the case of architecture 1, the optimal priority variables for the electric fan were:

- $\gamma_1 = 25.19\%$ (weight priority variable)
- $\gamma_2 = 0\%$ (diameter priority variable – ignored in this test case)
- $\gamma_3 = 0.037\%$ (cruise efficiency priority variable)
- $\gamma_4 = 74.78\%$ (peak power consumption priority variable)

The priority variables were used to parameterize the following objective function (replicated from appendix B).

$$\underset{X}{Min} \quad OEC(\overline{X}) = \gamma_1 \frac{m_{engine}(\overline{X})}{m_{engine-baseline}} + \gamma_2 \frac{diameter(\overline{X})}{diameter_{baseline}} + \gamma_3 \frac{1-\eta_{CZ}(\overline{X})}{1-\eta_{CZ-baseline}} + \gamma_4 \frac{PowEreq(\overline{X})|_{\max}}{PowEreq|_{\max-baseline}}$$

Note: η_{CZ} designates the overall efficiency of the propulsor in cruise condition and *PowEreq* the electric power required by the propulsor necessary to provide peak thrust requirements (which for this mission corresponds to the take-off/go around flight conditions).

The parameters noted "*baseline*" correspond to the attributes of the pre-sized electric fan (i.e. the first found feasible solution - see appendix B for more information). The pre-sized solution was defined based on the sizing thrust requirements listed in Table 41. The baseline attributes for the electric fan were:

- $m_{engine\text{-}baseline} = 4055$ kg
- $diameter_{baseline} = 329.8$ mm
- $\eta_{CZ\text{-}baseline} = 69.7\%$
- $PowEreq_{baseline}|_{\max} = 28.9$ MW

Therefore using the baseline attributes and the priority factors defines the objective function that should be used in subsystem developments:

$$\underset{X}{Min} \quad 6.212\times10^{-5}\left[kg^{-1}\right]\times m_{engine}\left(\overline{X}\right)-1.22\times10^{-3}\left[\%\right]\times\eta_{CZ}\left(\overline{X}\right)+2.592\times10^{-2}\left[\mathrm{MW}^{-1}\right]\times PowEreq\left(\overline{X}\right)\Big|_{\max}$$

In order to illustrate the meaning of this objective function, we can consider its meaning in terms of equivalence for subsystem trade-offs. Based on this objective function we can say that if the efficiency of the electric fan improves by 1%, it will be equivalent to a 19.7 kg weight improvement. Similarly, an increase in electric power requirement of 1kW in the peak power scenarios (take-off/go around flight conditions) is equivalent to gaining 0.4 kg. The ensemble of these trade-off factors is shown in the following table.

Table 42: Trade-off equivalences for the electric fan

	Δm_{engine}	$\Delta\ \eta_{CZ}$	$\Delta PowEreq$
Δm_{engine}	1	19.7 [kg/%]	417 [kg/MW]
$\Delta\ \eta_{CZ\text{-}baseline}$	0.0508 [%/kg]	1	21.2 [%/MW]
$\Delta PowEreq$	2.40 [kW/kg]	4.72 [MW/%]	1

As was observed in experiment 3.1, these equivalence factors are only valid for trade-offs involving designs similar to the one considered in conceptual design. Therefore, the values above are only valid for electric fans with attributes similar to the optimal sized solutions returned by the model:

- m_{engine} = 2871 kg
- η_{CZ} = 72.3 %
- $PowEreq|_{\max}$ = 25.5 MW

The trade-off factors presented in Table 42 are very useful for detailed developments. For instance, the electric fan described by the attributes above corresponds to the baseline, these trade-off factors will help the subsystem designer make decisions

with regards to his/her alternatives. For instance, let us consider a situation where a designer is considering the possibility to integrate an element which will improve efficiency by 1.5% in cruise conditions but which will compromise weight by 20kg. Using the objective function (or trade-off equivalences), the designer will know for sure that this addition, despite its weight penalty, will contribute to the optimization of the system as a whole.

6.6.4.2 Discussion of the Automated Subsystem Problem Formulation

The subsystem centric modeling approach allows for a practical symmetry between the modeling approach and the formulation of subsystem design problems. Since the models are formulated in a fashion that mimics subsystem developments, the information defined for their input can be used directly for the preparation of future developments.

The example above has demonstrated the ability of the methodology to translate the architecture design problem into a subsystem design problem. The coordinated optimization method allows for the formulation of an objective function which defines optimization priorities for the subsystem development. This formulation is very practical for subsystem developments because it is expressed in terms that pertain directly to subsystem attributes. This form is particularly practical because it requires no information external to the subsystem or further analysis in order to relate subsystem attributes to the architecture fundamental objectives. This formulation is particularly useful in a context where subsystem developments are outsourced to a tier company. In that context, the information that can be provided may be limited for intellectual property reasons. Using trade-off equivalences (in the form provided in Table 42) or using objective functions provides an effective and well packaged means for steering the optimization of subsystems. It is effective because it provides an unambiguous and quantitative formulation on how the subsystem should be optimized. It is well packaged because it

limits the necessary amount of information to a few parameters which are project specific (i.e. their validity is limited to specific subsystem solution and limited to context of integration) and can not easily reversed engineered (these are highly emergent factors resulting from the architecture analysis).

Based on these observations we can say that the methods provided in this thesis address research *Objectives 2* (*Definition of a strategic plan for the industrial development of the architecture*) and *5* (*Prepare a framework guiding detailed design developments*).

6.7 Conclusions on the Model-Based Architecting Methodology

This proof of concept has demonstrated several important capabilities provided by the Model-based architecting methodology. The resuling improvement to the state of the art can be organized in two major aspects:

- Setup of numerical analysis
- Exploration of the architectural potential

This proof of concept has unambiguously demonstrated the ability of the proposed methodology to improve the setup of architecture analysis models. This improvement is observable on two characteristic: the time to construct the analysis and the depth of the resulting analysis. The resulting analysis can capture many aspects of the necessary analysis of the architecture. It captures the sizing constraints implied by subsystem failures, but also by changes in the architecture configurations or in mission requirements. Implementing these levels of detail is essential to the sizing of the architecture. Generally, the implementation of architecture analysis requires weeks of effort. With the proposed methodology, this effort is reduced to a few hours (acceleration by a factor of 14). Consequently, it enables the accurate exploration of many architecture concepts in situations where only one or few concepts could be explored in the past.

The proposed methodology is also an enabler for the optimization of the subsystem sizing. This complex optimization process is very difficult to setup and often neglected in traditional conceptual activities. Based on the Coordinated Optimization method (discussed in more details in the previous chapter), this optimization is automatically setup. This important ability detects the true potential of each architecture concept considered. Without this optimization, subsystem sizings are suboptimal and the architecture analysis is concervative. In the proof of concept considered in this chapter, we have observed that sizing subsystems without optimizing the sizing priority factors would lead to conclusions up to 50% worst than the best value.

This unfair comparison will lead to an architectural status-quo (better but new architectures would not be considered because their optimization potential is not considered). New architecture concepts would be sized in a suboptimal fashion and compared to a baseline which has been optimized in previous programmes. We can therefore observe that the proposed methodology enables detection of architectural improvements which would otherwise remain untapped.

Chapter 7

Conclusions and Future Work

7.1 Overview of the Scientific Reasoning

This dissertation proposes a methodology which enables the conceptual analysis or aircraft system architecture. The proposed ideas (introduced in Chapter 4) were based on several hypotheses tested on both, a series of focused experiments verifying the validity of its constituting methods (Chapter 5), and a proof of concept (Chapter 6) to demonstrate the practicality of the overall process. An overview of the scientific reasoning underlying both the origin and validation of these hypotheses is proposed in this section. A graphical overview of the underlying motivation and validation of the hypotheses is also provided in Figure 239 (at the end of this section).

In order to quickly compare architectures, it is necessary to formulate the mission in a generic fashion. This need motivated the hypothesis that the mission could be parameterized using the flight phases, the flight conditions and failure criticality levels to organize the functional requirement imposed by the mission (*Hypothesis 1*). This technique was validated by its successful application to a typical commercial transport aircraft mission (section 5.1). In this experiment the proposed technique was able to capture the operation constraints implied by the mission, including requirements in failed conditions as specified by certification authorities (JAA&FAA)

The comparison of architectures implies that we are capable of sizing their subsystems based on the requirements implied by the mission. In order to maintain the necessary simplicity of sizing while keeping the ability to optimize subsystem solution, the optimizer-based sizing technique was proposed (*Hypothesis 3.1*). The need for this technique was validated by observing that a given set of requirements can always be

satistifed by more than one subsystem solution. Without a means to select the best solution, the sizing estimations at the subsystem-level are inaccurate. This observation discussed in section 5.2 allowed for the validation of *Hypothesis 3.1*.

The third set of hypotheses concerned the means to steer the subsystem sizing toward a global optimum rather than a subsystem optimum. The Coordinated Optimization technique was proposed and supported by *Hypothesis 3.2*. This technique is a multi-level optimization where an architecture-level optimizer coordinates the subsystem optimizationers toward architecture-level goals by changing their objective functions. The validity of this hypothesis was supported by the comparison of a direct optimization method to the coordinated optimization. This comparison showed that the Coordinated Optimization is capable of identifying the global optimum solution for different types of design problems (section 5.3).

The Coordinated Optimization identifies best subsystem solutionsby searching for the best optimization priorities for each subsystem. *Hypotheses 3.3* and *3.4* propose using these optimization priorities to formulate better requirements for future subsystem developments. These hypotheses were supported by a mathematical analysis and a statistical analysis of simulated developments. The mathematical analysis demonstrated the theoretical validity of the method. The simulation of developments demonstrated on a series of test problems the ability of the method to improve the final architecture performance after subsystem developments. Based on experiments shown in section 5.3.5, using this method allows improving the performance of the developed architecture in 95% of cases and increases the probability of meeting performance levels by up to 25%.

Based on these techniques, a model-based architecting methodology was proposed. This methodology enables the definition of architecture concepts for which executable architecture analyses are automatically constructed. Each analysis is constructed by assembling subsystem modeling bricks to represent a specific architecture concept. These assembled analyses estimate and compare the performance of

architectural alternatives. The building method and process are based on *Hypothesis 2* which was validated by applying the methodology to a proof of concept. The proof of concept is presented in Chapter 6 and describes the architecture trade-offs for a distributed turbo-electric propulsion concept for a commercial passenger aircraft. This experiment has demonstrated that the method is capable of rapidly generating accurate executable analyses. This accuracy is supported by the ability of the executable model to capture failed operating conditions and variations in mission requirements associated with each architecture concept. The estimated acceleration of the time necessary to setup each analysis was reduced by a factor of 14. Also the proof of concept has highlighted the ability to automatically implement the optimizer-based sizing and coordinated optimization techniques in the executable analysis. As a result, the model-based architecting methodology allows for the rapid construction of deep and architecturally-accurate analyses.

It can therefore be concluded that the proposed methodology allows for the exploration of a multitude of architectural alternatives. Previously, these trades were limited to few and small variations from an existing architecture baseline. Using the model-based architecting methodology, radically new architecture concepts can be explored and sized accurately. The rapidity of the architecture analysis model generation allows for exhaustive exploration of the architectural design space. The resulting analysis models are based on advanced sizing and optimization techniques which not only allow for accurate architecture sizing and synthesis but also provide an improved understanding on the subsystem context of integration. These techniques explore the subsystem-level trade-offs and allow for an effective steering of subsystem developments toward architecture-level goals.

A graphical version of this overview is provided in Figure 239.

Figure 239: Overview of the scientific reasoning

7.2 *Overall Conclusions on the Methodology*

This thesis attempts to address the challenges associated with architectural innovation. This technical gap must be addressed in the conceptual design phases. The difficulty associated with this task lays in an intricate intersection between the field of system engineering (due to the concept scale and complex nature of the problem) and the field of common engineering due to the technical complexity of its analysis. In order to address this gap, the methodology presented below was proposed.

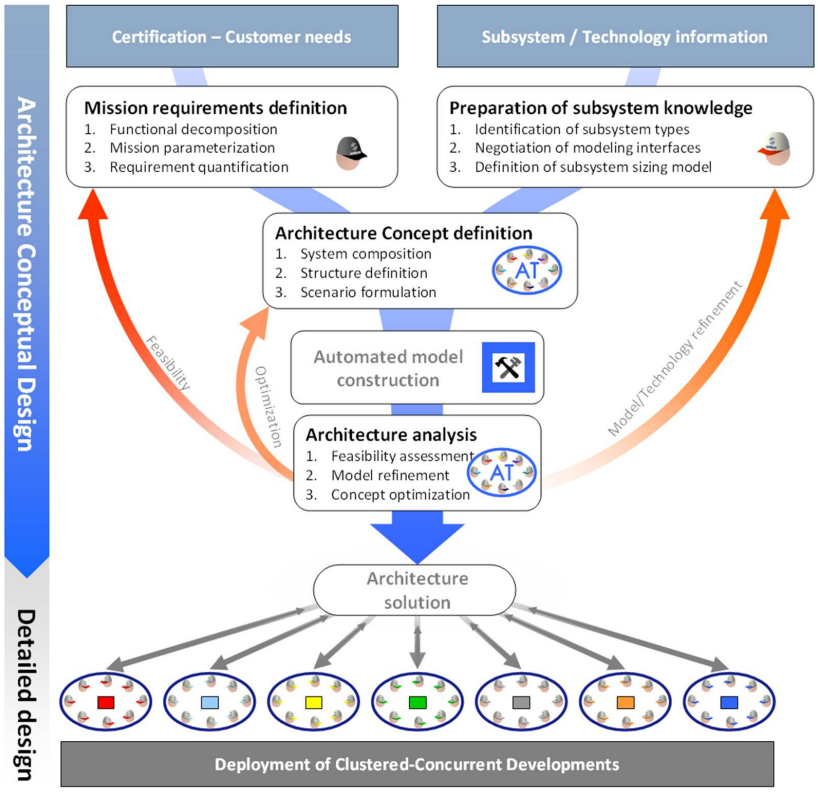


Figure 240: Overview of the proposed architecting process

7.2.1 **Complete Architecting Process**

The methodology proposed in this thesis provides a complete concept architecting process. It includes a convenient formulation of the requirements which is valid regardless of the architecture concept to be investigated. The process integrates the subsystem knowledge in both a flexible and an effective fashion. This approach captures

arbitrarily complex subsystem information via the preparation of modeling bricks with standardized interfaces. In order to describe the architecture concepts, the methodology provides a simple, effective and visually unambiguous method using SysML. Based on this definition, the test-case has demonstrated the ability of the proposed model construction method to translate the SysML concept description into an executable analysis model. Hence, this methodology allows for the analysis of the architecture attributes and supports numerous forms of trades and architecture exploration activities.

7.2.2 **Provision of Deep and Flexible Architecture Analyses**

The nature of the resulting analysis model allows for a deep, bottom-up analysis of the architecture. The automatically generated analysis provides unprecedented flexibility in modeling. What used to take months of preparation can now be prepared within a few hours. The quality and depth of the analysis does not even compare with traditional system engineering techniques (QFD, change propagation and the like). Therefore, this methodology is able to provide better analyses faster. These important aspects are key enablers for the exploration of new architecture concepts and the detection of new architectural opportunities.

7.2.3 **Exploration of Subsystem Development Strategies**

Beyond the conceptual architecting abilities provided by the methodology, the Coordinated Optimization method provides a powerful means to explore the optimization priorities for subsystem development. This optimization technique allows for both effective and optimized subsystem sizing estimations, but it also detects the relative importance of subsystem attributes from a system-level perspective. This detection translates architecture-level optimization objectives into a subsystem-specific objective formulation. The identification of such information translates the aircraft vision (or objective), into technically clear and unambiguous optimization plans that can be

implemented in subsystem developments. For instance, let us say that we want to mimize the take off weight. This aircraft level objective implies minimizing both the engine weight and/or the fuel weight necessary to perform the mission. Naturally it is difficult to do both simultaneous. By informing the engine designer what trade-off threashold he/she can use for the weight and efficiency optimization, the methods translates the system (or aircraft) vision into an optimization strategy at within the subsystem context.

7.2.4 **A Collaborative Framework**

It is important to note that in this process, the subsystem experts are involved in almost every step. The information which supports architectural decisions is constituted by subsystem expert knowledge. They participate collaboratively in the definition of the architecture concept; therefore, they are invested in the design effort. The analysis of results is observed both at the architecture and subsystem-levels. The subsystem experts are responsible for checking and monitoring the subsystem attributes. Their deep involvement in architectural trades enables them to relate their future subsystem developments to the architecture-level vision.

Therefore, the expected outcome of this involvement is the decentralization of the understanding of the architecture concept necessary for successful implementation of future developments. Consequently, these experts will be able to conduct their design activities with a stronger aircraft-level perspective and better understanding of inter-subsystem constraints.

7.3 Future Work

The work presented in this thesis should motivate further research in the area of model-based system architecting. In the preparation of this document, great effort was dedicated to the documentation of the work and tools supporting the methodology. This documenting effort was gladly spent in the hope that it will be useful in the future.

7.3.1 **Development of Monetary Objective Functions for Subsystem Developments**

One of the main contributions of this method is the translation of architecture level objectives into subsystem optimization priorities. In the test cases presented in this thesis, the subsystem optimization priorities were formulated as abstract pondered sums of key subsystem attributes. This abstract form is difficult to interpret and can be confusing for the novice. Therefore, the field would greatly benefit from formulating subsystem-level objectives in a monetary form. Instead of trading weight in kg for power demand in kW, everything would be defined with respect to cost impact in Euros or Dollars on the overall project. Having a monetary metric for steering subsystem developments would be a very powerful means to plan for internal research and development efforts and negotiate outsourced development contracts.

7.3.2 **Methodological Augmentation of Model-Based Architecting Methods**

The aircraft system architecture includes many redundant subsystems. Often these redundant subsystems are symmetrically defined (they all share the same design). This symmetry allows for the simplification of the analysis necessary for the sizing process. This simplification is not yet captured by the methodology proposed in this thesis.

The methodology was implemented on aircraft power system architectures which include very different types of functional relationships. It would be very interesting to proceed toward the implementation of other key system integration domains like system

installation and heat management. The intricate inter-relationships between these domains could be a fascinating proof of concept for demonstrating (or further testing) the ability of the methodology to adapt to a broader domain than the one presented in Chapter 6.

7.3.3 **Capturing Transient Behavior**

In the examples provided in this thesis, the functional relationships were defined based on quasi-static requirements and specifications. Using quasi-static qualifications of interaction may not (under some circumstances) capture the actual sizing constraints. This is particularly true for some electric power systems which get sized based primarily on their transient behavior.

It is possible to capture these transient behaviors while maintaining a quasi-static formulation of the problem using transient boundaries (as presented in the interface standard by the United States Department of Defense [74] and implemented by Khozikov et al. in reference [75] and Phan in his PhD dissertation [76]). Therefore, instead of sizing only on maximum (and nominal) power requirements, the subsystem is provided with a maximum possible envelope of power or specification fluctuations which can be used to capture the most critical transient behavior. Having the ability to provide such qualification would greatly improve the flexibility of the proposed method and would facilitate improved model accuracy.

7.3.4 **Visualization of Analysis Results**

The methodology has presented a practical and visual means for the formulation of the architecture concept. This means facilitates the collaboration between the experts by presenting the information in a visually intuitive fashion. On the other hand, the results produced by the model are still presented in a obscure way, which is difficult to understand and the information is sometimes difficult to access.

Since the (SysML) conceptual model already provides a great visual context, the effectiveness of the methodology would be greatly improved by displaying the information defined in the analysis model back into the conceptual model definition. This new feature would greatly facilitate the optimization of the architecture concept, because the location of the information would hence correspond to the location where new architecture concepts are defined. Notional examples are provided in the following figures.

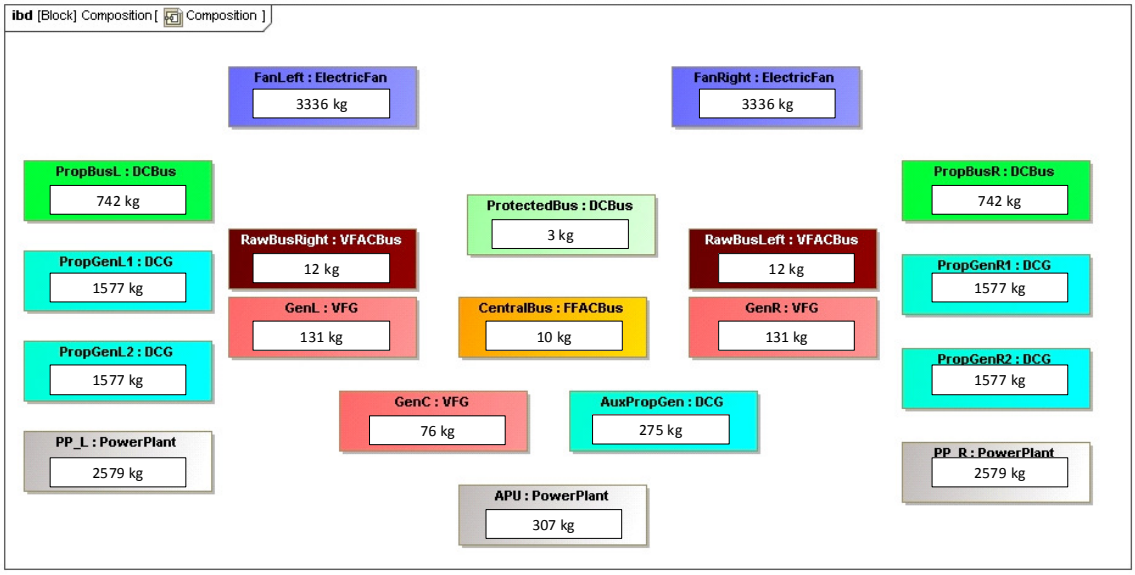


Figure 241: Example Displaying Subsystem Attributes

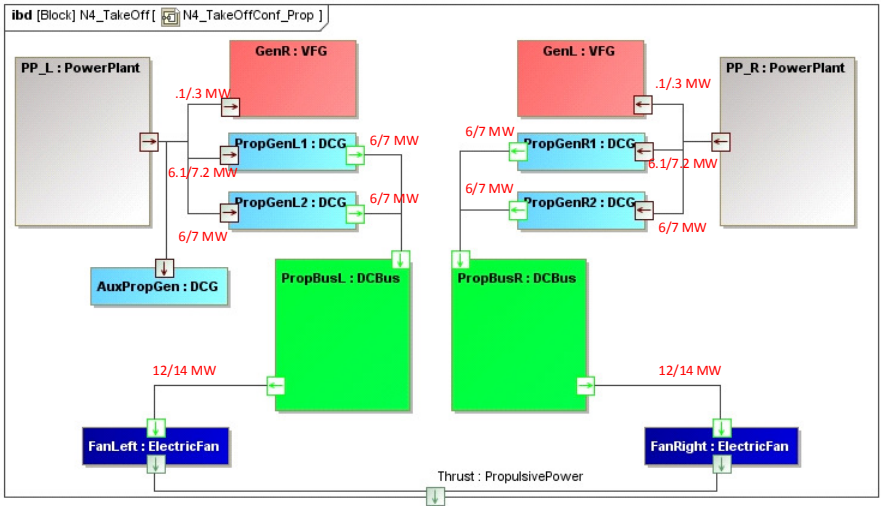


Figure 242: Example Displaying Functional Flow Attributes

7.3.5 **Industrial Implementation**

Since its genesis, this methodology has been developed with a strong emphasis on the industrial context and reality. The ultimate objective of this thesis is to see this type of work supporting the current development activities within the European Aerospace Defense and Space company (including Airbus S.A.S.) which has been supporting this project financially and technically. The large product portfolio of this organization offers many fascinating opportunities to implement the architecting methodology proposed in this thesis. The obvious application is in commercial aircraft development (as presented by the test case presented previously). But the methods presented in this dissertation can be applicable in fields well beyond aircraft design.

This methodology enables the integration of diverse point of views and expertise and the translation of a conceptual model into an executable analysis model. These enablers are of common interest to a large spectrum of complex system developments. Therefore, other forms of complex system analysis could be interested in the methodology. Some possible applications could be other forms of airborne vehicles (rotorcraft, military vehicles in general), land based vehicles (trains, automobile) or maritime vessels. In a different fashion, the methodologies associated with this thesis could be useful in the Systems-of-Systems field.

7.4 Final Words

The field of model based system architecting is bound to grow in the future. We live in a world where the technical limits are pushed a little further every day. But as we progress technically, the complexity of the world we live in grows in proportion. System engineering and architecting methods provide us with the capability to master and manage this ever growing complexity. They are our main hope to sustain progress in the future. I hope that the reader will believe that this dissertation has been a positive contribution in addressing this ever lasting challenge.

Appendix A

Quantification of boundary function specifications

A.1 Introduction

The architecture considered in this thesis is the power architecture of a commercial passenger transport aircraft. In this chapter we shall define requirements for the power architecture. These power requirements are based on the aircraft boundary functional requirements. These boundary functions define what the aircraft is expected to perform (e.g. take-off within a specified distance, offer in-flight-entertainment, etc…). These requirements are translated into the requirements they imply on the power architecture. This translation implies that if the power architecture is able to provide its specified requirements, the aircraft will be able to perform the vehicle level requirements.

As prescribed by the methodology chapter of the thesis, the vehicle level requirement relevant to the power architecture were broken down and organized by functions. The functions assigned to this architecture are to provide the propulsive and non-propulsive power necessary for the aircraft to perform its mission. This appendix presents the value used to quantify the power requirement implied both by propulsive and non propulsive functions.

The following section of this appendix describes the origins of the values quantifying the requirements. This description will present the references and assumptions behind them. The following sections will be organized around the following topics:

- Aircraft mission general description
- Thrust requirements
- Customer and technical loads
- Wing icing protection

- Control
- Environment conditioning

A.2 Thrust requirements

The thrust requirements are derived from vehicle performance requirements. Therefore this section will first describe, vehicle-level requirement. These requirements must then be translated from aircraft-level performance metric into propulsive requirements. The analysis supporting this translation will then be presented in the firtst part of this section. The second will present a calculator which was develop to perform the translation of vehicle-level performance requirements into propulsive requirements. Then this section will be concluded by explaining how the various certification and customer performance requirements were mapped to the flight phases and used to define the thrust requirements.

A.2.1 Vehicle performance requirements

These requirements are either certification rules or other targets exceeding the certification requirements (motivated by customer/stakeholder needs and preferences). These vehicle performance requirements specify manoeuvres that the aircraft is expected to be capable of performing. This section will describe these vehicle level requirements along with their origins and/or motivation. The final paragraph of this section will provide a reflection on the applicability of JAR/FAR certification requirements to distributed electric propulsion.

Joint Aviation Authority and Federal Aviation Administration regulations

In this work, the certification rules considered were the Joint Aviation Regulations (JAR) and the Federal Aviation Regulations[5] (FAR). The section relevant to commercial transport aircraft are the FAR part 25 and JAR part 25. The requirements driving the thrust were primarily those setting a minimum climb requirement for the aircraft under various scenarios. These requirements are listed in the following table:

Table 43: JAR/FAR 25 climb requirements

		Number of engines				
	Eng. Out	4+	3	2	Climb type	JAR/FAR ref.
TO seg1 (with eng. Out)	1	0.5	0.3	0	rate	25.121
TO seg1 (all eng. Op)	0	1.7	1.5	1.2	rate	25.111
TO seg2	1	3	2.7	2.4	rate	25.121
Final TO seg	1	1.7	1.5	1.2	rate	25.121
Approach segment	1	2.7	2.4	2.1	rate	25.121
Aborted LD (with eng. Out)	0	3.2	3.2	3.2	rate	25.121

The manoeuvres associated with these requirements are described by regulations. The following table provides an interpretation of the JAR/FAR manoeuvre descriptions. This table was used to setup the analysis presented in part A.2.2 of this section:

[5] The JAR are specified by the European-based Joint Aviation Authority and the FAR by the US-based Federal Aviation Administration

Table 44: Manoeuvers implied by JAR/FAR

First take-off segment	Flight condition:	After lift-off (no ground effects) about take-off altitude Max take-off gross weight (TOGW)
	Engine operation:	0 or 1 engine failed Surviving engines at take-off regime
	Configuration:	Wing in take off configuration (flap extended) Landing gear down
Second take-off segment	Flight condition:	400 ft above take-off altitude TOGW
	Engine operation:	(Idem first segment)
	Configuration:	Wing in take off configuration (flap extended) Landing gear up
Final take off segment	Flight condition:	FL 150 Approximately at TOGW
	Engine operation:	1 engine out Surviving engines at maximum continuous regime
	Configuration:	Wing in take off configuration (transition to enroute configuration) Landing gear up
Approach segment	Flight condition:	150% of Vstall Maximum landing weight
	Engine operation:	1 engine out Surviving engines at take-off thrust
	Configuration:	Partial landing configuration (flap/slat not fully deployed Gear up
Aborted landing	Flight condition:	130% of Vstall
	Engine operation:	All engine operating All engines at take-off thrust regime (regime considered: 8 sec after throttling up from idle)
	Configuration:	Full landing configuration Landing gear down

Other performance requirements

In addition to these certification imperatives, several requirements were considered. These additional requirements will either set sizing constraints more constraining than those specified by certification or will bring additional information on the context of operation.

Note: Some of the requirements below may be controversial and a little “sophisticated”. But some of these requirements (especially the short take-off procedure) were included to highlight the ability of the methodology proposed in this thesis to

- Capture complex requirements (i.e. requirements conditioned by failure scenarios)
- Size the aircraft in a fashion that focuses on performance at nominal operations while ensuring survival in failure conditions.

Take-off operations

The first requirement concerns a noise reduction procedure for take off. Several projects have proposed take-off procedures narrowing the noise footprint of the aircraft on surrounding communities. These procedures imply that the aircraft takes off over a very short distance and performs a first take-off-climb segment with a steep gradient. This procedure allows the aircraft to concentrate the generation of noise within the airport property and to reach higher altitudes once the aircraft flies over the nearby communities. To estimate the thrust necessary to realize these procedures, the static thrust requirements imposed in (flight phase 4) will be calculated for a Short-Take-Off Field Length (STOFL) using 75% of the distance corresponding to an standard A320 (namely 1725 m instead of 2300 m). Also the climb gradient the take-off-climb segment 1 will be of 15%. These manoeuvres will only be enforced for no or dispatchable failure situations (not failures with criticality levels 2 or 3). For failure scenarios, the current A320 take off performance and certification climb gradient will be used to define thrust requirements. This procedure, meant to reduce the noise footprint, implies that the take off manoeuvre will be operated on a regular runway. Therefore the initial climb segments occur above the runway – concentrating the noise emission within the airport vicinity) and in case of critical propulsive failure, the procedure can be reverted to the standard take-off using the full length of the runway.

Climb acceleration

It is assumed that the climb is performed in two phases. The first phase goes from take off to the altitude of 10,000 ft. As part of this phase the aircraft is expected to be able to sustain the second segment take-off climb requirements below 1500 ft altitude and final take-off climb to 10,000 ft. The ability to climb with all engines operating is not regulated for all engines operating. Therefore targets of 7.5% and 5% were respectively

used for the second and final take-off climb segments. In addition to these climb gradient targets, the following excess power requirements were imposed:

Table 45: Excess power requirements

Altitude	Nominal	Level II	Level III	
FL<100	**1500**	**700**	**500**	[ft/min]
FL>/=100	**500**	**400**	**300**	

Certification for distributed electric propulsion

Certification rules are defined to provide a generic standard by which aircraft manufacturer can certify that their vehicle is sufficiently safe for operation. Formulating a generic standard is essential to guaranty the "fairness" of the requirements to all applicants (despite their differences in design concepts and approaches) while maintaining the safety standard. But the "critical engine inoperative" configuration, mentioned in several certification requirements, is unclear with respect to its application to the distributed electric propulsion architecture. For example let us consider the FAR requirements for the first take-off-climb segment (extract from FAR section 25.121):

> Takeoff; landing gear extended. In the critical takeoff configuration existing along the flight path (between the points at which the airplane reaches VLOF and at which the landing gear is fully retracted) and in the configuration used in §25.111 but without ground effect, the steady gradient of climb must be positive for two-engine airplanes, and not less than 0.3 percent for three-engine airplanes or 0.5 percent for four-engine airplanes

In traditional architectures, this configuration textually refers to situations with a failure with a turbofan, turboprop or turbojet. Distributed electric propulsion architectures do not include such system. However the absence of a turbofan, turboprop or turbojet does not address the safety issue targeted by the certification requirement. In situation of propulsive failure, the aircraft must still find a way to climb safely. Therefore, we can see that the certification requirement is not related to the engine itself but its original functions in the traditional architecture.

Therefore in order to generalize the degraded requirement in degraded context, let us analyze the extract from FAR 25.121 presented above. In the paragraph cited above, the requirement is defined as a function of the number of engine in the architecture (2, 3 or 4) and specifies that the architecture is in its "critical takeoff configuration", which implies that one engine is inoperative (Raymer [87] appendix A). If we consider that N_{eng} is the number of engine onboard and that n_{fail} is the number of failed engine, we can define the degree of availability k_{avail} of the architecture capability to provide the function "Propel the aircraft":

(54)

$$\kappa_{avail} = \frac{N_{eng} - n_{fail}}{N_{eng}}$$

Following this method Table 43 was translated into the following table:

Table 46: Degraded requirements from the FAR/JAR

Engine conf (nf/Neng)	0/4	0/3	0/2	1/4	1/3	1/2
kavail	100%	100%	100%	75%	67%	50%
TO seg1	1.7	1.5	1.2	0.5	0.3	0
TO seg2				3	2.7	2.4
Final TO seg				1.7	1.5	1.2
Approach segment				2.7	2.4	2.1
Aborted LD	3.2	3.2	3.2			

In distributed electric propulsion concepts the propulsive functionality is performed by three layers of physical elements:

- The systems dedicated to the boundary function of propelling the aircraft (ducted electric fans)
- The power subsystem dedicated to the functions of transforming and distributing the energy dedicated to the electric fans (including propulsive power buses and generators)
- And power plant subsystems dedicated to the function of providing mechanical energy to the propulsive electric generators (gas turbines)

If we were to going to follow certification rules textually, we could interpret the failed engine cases as situations were a ducted electric fan failed. But clearly (and even if we had only 2, 3 or 4 electric fan), the original motivation beyond the certification rule would not be adequately represented by the failure of one failed ducted fan. Indeed, what would be the tolerable impact of failed propulsive generators or power plants? Therefore instead of a textual application of the certification rules, the degraded thrust requirements are indexed on the k_{avail} factor implied by the certification texts.

One of the methods proposed in this thesis classifies failures by criticality levels. It is important to note that this classification is based on how much of the functional capability is available. The level II was defined by a failure which affects less than 50% of the propulsive capability while the level III failure implies a failure affecting 50% or more. As a result, the thrust requirements associated with level II configurations is determined using certification requirements implying a k_{avail} greater than 50% while level III requirements will be based with a k_{avail} less equal or less than 50%. Based on this approach, the following requirements were selected for the degraded performance requirements:

Table 47: Requirements degradation profiles for study

	Requirements used			
	Nominal	Level II	Level III	
TO seg1	10	0.5	0	Climb gradients
TO seg2	7.5	3	2.4	
Final TO seg	5	1.7	1.2	
Approach segment	5	2.7	2.1	
Aborted LD	3.2	2	0	

Note: The number represented in blue in the table above represents customer requirements which are independent from JAR/FAR rules.

The results above were defined based on the certification requirements presented in Table 46. They are presented together in Figure 243.

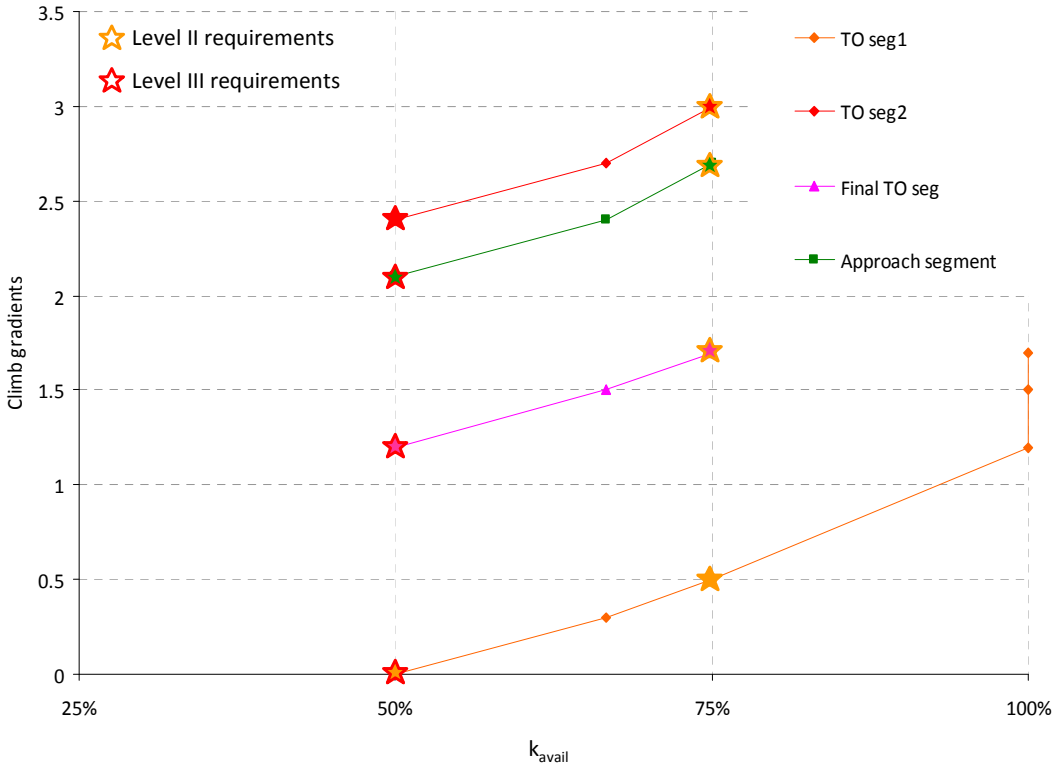


Figure 243: Degraded requirements from the FAR/JAR

A.2.2 Deriving thrust requirements from vehicle performance requirements

Relationship between thrust and vehicle performance

The relationship between thrust, manoeuvring characteristics and performance can be obtained by the commonly used "energy equation" by Mattingly[25]. Since in this design problem, the drag is broken down in different fashion and the wing loading is fixed, a similar but adapted form of the energy equation was derived. This approach is based on the same assumption as the Mattingly energy equation namely:

- steady state operation
- the vehicle is considered to be a point mass
- thrust and velocity are oriented in the same direction
- the climb angles θ are assumed to be small so that sin(θ) ~ θ

Based on these assumptions, the change in energy is defined as:

(55)

$$(T-D)\times V=\frac{d}{dt}\left(m\times g\times h+\frac{1}{2}m\times V^2\right)$$

$$\text{Since } W=m\times g \text{ then } T=D+\frac{W}{V}\frac{d}{dt}\left(h+\frac{1}{2}\frac{V^2}{g}\right)$$

The thrust formula can be reformulated using:

- Excess power, defined as $Ps=\frac{d}{dt}\left(h+\frac{1}{2}\frac{V^2}{g}\right)$
- Drag formulated as $D=q\times C_D\times S$, with q referring to the dynamic pressure $q=\frac{1}{2}\rho V^2$.

The drag coefficient is a function of:

- the design of the aircraft and its configuration
- the lift coefficient CL
- the mach number
- the air properties (altitude)

By substituting the term introduced above, equation (55) can be reformulated as:

(56)

$$T=q\times C_D\left(C_L,MN,Alt,Configuration\right)\times S+\frac{W}{V}\frac{d}{dt}Ps$$

The load factor n defines the lift as a function of the aircraft weight at the time the manoeuvre is executed W: $L=n\times W$. The lift coefficient is defined as:

$$L=q\times C_L\times S$$

Therefore CL can be defined as:

(57)

$$C_L=\frac{n\times W}{q\times S}$$

Representation of aerodynamic forces

In this experiment the designs of the wing (surface area, aspect ratio, taper ratio, airfoil profile, sweep, etc…) and the airframe are fixed. The wing configuration, however, is dependent on the manoeuvre (i.e. flight phase). The three wing configurations considered were:

- the take off configuration
- the landing configuration
- and the enroute configuration

Their aerodynamic characterization is described in the following paragraphs.

Dirty configurations (take off and landing aerodynamic performance)

The take off and landing configurations referred to as dirty configurations refer to the wing with flaps and slats deployed to different degrees. The landing configuration generally refers to a configuration were flaps are fully extended while the take-off configuration implies a slightly lower degree of extension.

The dirty configurations are generally used on a relatively narrow range of Mach number and altitude. Therefore the representation of the Mach number effects and altitudes were not considered in the function provided for the drag which followed the form represented by equation (58):

(58)

$$C_D(C_L) = K_1 \times C_L{}^2 + K_2 \times C_L + C_{D0} + (C_{LG})$$

To determine this equation, the polar curves used for the two dirty configurations by the EDS group were considered. By performing a least square error regression, terms K_1, K_2 and C_{D0}, were determined. The original data and the resulting polar curve for both configurations are represented in the following figure.

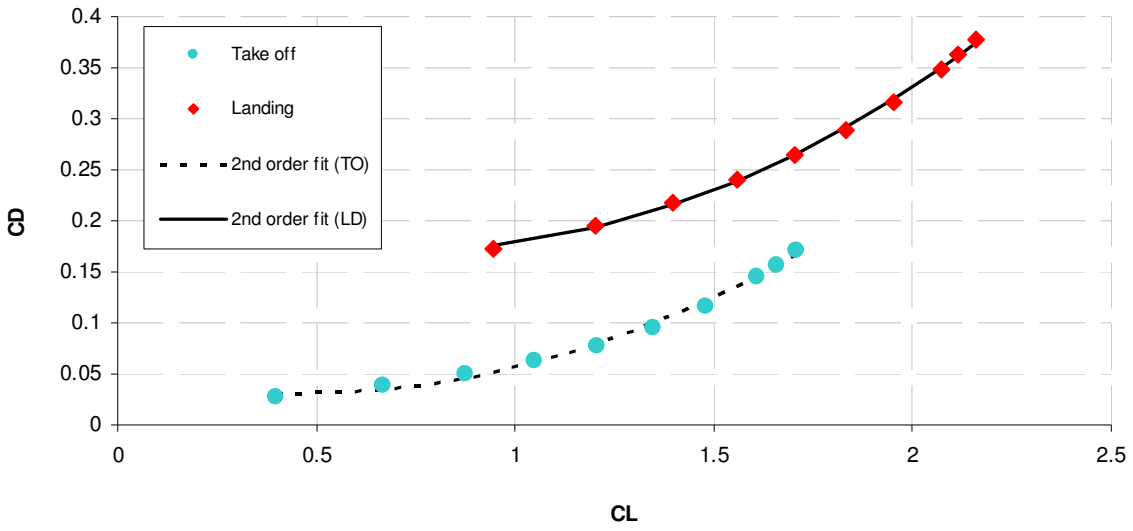


Figure 244: Polar curves for the dirty configurations

The values for term K_1, K_2 and C_{D0} are listed in the following table:

Table 48: Dirty configuration factors

	TO Conf	LD conf
K1	0.087517	0.0978955
K2	-0.081174	-0.140485
Cdo	0.0497961	0.2203576

In addition to these three terms, and additional term was used to represent the drag produced by the deployed landing gear. The value used for this term was 0.005. This term was kept separate from C_{D0} as the landing gear is not always deployed simultaneously with landing or take-off configurations.

Clean or enroute configuration

The aerodynamic characterization used herein is similar to the one used by FLOPS. The drag coefficient characterized as follows:

(59)

$$C_D(C_D, M, Alt) = C_{D-f}(M) + C_{D-comp}(M) + C_{D-P}(C_L, M) + C_{D-i}(C_L) + \Delta C_D(M, Alt)$$

$C_{D\text{-}f}$ represents the friction drag, and $C_{D\text{-}comp}$ the compression drag. Both these coefficients are only a function of the Mach number. The pressure drag $C_{D\text{-}p}$ is a function of the Mach number and angle of attack (i.e. C_L). $C_{D\text{-}I}$ is the lift induced drag. ΔC_D makes

the necessary adjustments to capture other variations resulting from changes in air properties (operating speed and altitudes).

To evaluate each of these drag coefficients the calculator performs an interpolation on data provided by five tables representing each term in equations (59). These tables were developed by the EDS teams to represent an A320.

Analysis associated with take-off distances

One of the requirements mentioned above concerns take-off field length. The analysis presented above does not directly apply to the estimation of distance and requires a slightly different starting point. This form is similar to the one proposed by Raymer [87], Mattingly [25] and Anderson [8]. If we consider the definition of the term of acceleration and Newton's First law:

(60)

$$\vec{a} = \frac{d\vec{v}}{dt} \text{ and } \vec{a} = \frac{\sum \vec{F}}{m}$$

By introducing the term $d\vec{s}$ for change in position equation becomes:

(61)

$$\vec{a} = \frac{d\vec{v}}{dt} = \frac{d\vec{v}}{d\vec{s}}\frac{d\vec{s}}{dt} = \vec{v}\frac{d\vec{v}}{d\vec{s}} = \frac{\sum \vec{F}}{m}$$

During ground acceleration all forces and displacements are occurring in a path parallel to the runway tarmac. (The sum of lift and normal reactive force from the ground are assumed to be equal). Therefore the vector form of equation (61) can be formulated in a scalar form:

$$ds = \frac{m \times v}{\sum F} dv \text{ with } \sum F = T - D - R$$

T is the combined thrust of the propulsive systems, D is the drag produced by the airframe and wings and R is the resistive force on the landing gears. They are defined as

(62)

$$T = T_0 \times \alpha(M_a) \quad D = 1/2 \times \rho \times v^2 \times S \times C_{Do} \quad R = \mu \times W$$

Notes: μ is the resistive coefficient of the landing gear. The drag formulation above assumes that no lift is produced. The amount of thrust that can be produced during the acceleration process may change significantly for a ducted fan. At low altitude, the electric ducted fan will typically be running at the motor power limit (unless the motor has oversized or the fan undersized). In order to capture the variation in thrust due to this limit, a lapse function was determined so that the take-off distance can be translated into a sea-level-static thrust requirement. The lapse function was defined as the following equation based on a least square regression on data provided by the ducted electric fan performance model described in Appendix B.

(63)

$$\alpha(M_a) = 0.9054 \times M_a - 1.5469 \times M_a + 1$$

By including expression in equation (62), the distance "s" necessary to go from speed 0 to some speed "V" can be formulated as follows:

(64)

$$s = \int_0^V \frac{m \times v}{T_0 \times \alpha\left(\frac{v}{a_a}\right) - 1/2 \times \rho \times v^2 \times S \times C_{Do} - \mu \times W} dv$$

To evaluate this integral, it is possible to perform a step-wise resolution of the form:

(65)

$$s = \sum_0^N \frac{m \times v_i}{T_0 \times \alpha\left(\frac{v_i}{a_a}\right) - 1/2 \times \rho \times {v_i}^2 \times S \times C_{Do} - \mu \times W} \Delta v$$

$$\text{With } \Delta v = \frac{V}{N} \text{ and } v_i = \frac{i \times V}{N}$$

Overview of the take-off procedure

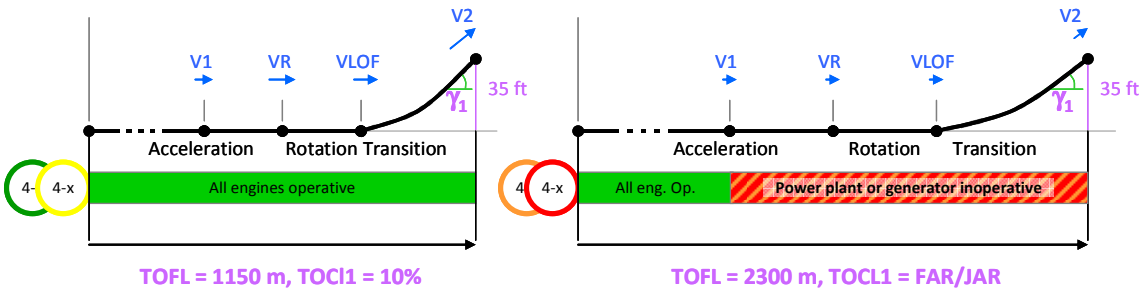


Figure 245: Take-off sequence

The **take-off sequence** is constituted of 6 phases or segments:

- Acceleration pre-V1: The aircraft is set in motion with engine operating at full power. If an engine failure was to occur at any point in this segment the take-off sequence can be aborted. This segment ends once V1 is reached. V1 is referred to as the decision speed.
- Acceleration post-V1: The aircraft continues its acceleration toward its rotation speed. Regardless of the engine failure which may potentially occur after this point, the aircraft has to finish its take-off sequence (too late to brake and stop).
- Rotation: The aircraft has reached VR. VR is the rotation speed at which the elevator becomes sufficiently effective to pitch the aircraft up to a sufficient angle to lift-off.
- Transition (Take-off-climb segment 1): The aircraft gets airborne and climbs up to a ground altitude of 35 ft. The speed at this point is V2. A climb gradient is specified by FAR/JAR 25.121 (see Table 44).
- Take-off-climb segment 2 and final take-off-climb are the climb sequences where the aircraft morphs from its take-off to enroute configuration. For each of these segments climb gradients are specified by regulation (see Table 44).

For measuring the take-off distance, the FAR/JAR 25.113 consider the distance between the stand-still position and the point where the aircraft reached 35 ft ground

altitude. This distance is measured for a take-off sequence where all engines are operating until V1 is reached, then with the "critical inoperative configuration" for the rest of the sequence. This distance is also referred to as the **balanced take-off field length**.

In the requirements described above two sets of requirements were specified for the take distance. The first one corresponds requires the aircraft to perform a low noise footprint take off procedure. This requirement is only enforced to non-failed cases. The second requirement is a degraded functional requirement which focuses on the safety of the take off procedure.

Each thrust procedure will require different amount of thrust and will be constraining in different ways. Therefore two take-off analyses will be used to translate the take off requirement into thrust requirements. One will correspond to the short take-off field length (STOFL) for the narrow footprint procedure, and the other to the Balanced take-off Field Length (BFL) which correspond to failed engine cases.

Estimating the field length necessary to the acceleration segment

In the estimation of the take-off field length, all engines are operating (both ducted fans and power plants). Using the distance equation (64), the equation for the acceleration distance corresponds to:

(66)

$$S_{acc} = S_{acc1} + S_{acc2} = \int_0^{V_R} \frac{m \times v}{T_0 \times \alpha\left(\frac{v}{a_a}\right) - 1/2 \times \rho \times v^2 \times S \times C_{Do} - \mu \times W} dv$$

For the BFL requirement, the thrust available in each of the two phases of acceleration are different. Therefore the integral has to be performed in a step-wise fashion.

(67)

$$S_{acc1} = \int_{0}^{V1} \frac{m \times v}{T_0 \times \alpha\left(\frac{v}{a_a}\right) - 1/2 \times \rho \times v^2 \times S \times C_{Do} - \mu \times W} dv$$

$$S_{acc2} = \int_{V1}^{VR} \frac{m \times v}{T_{0\,fail} \times \alpha\left(\frac{v}{a_a}\right) - 1/2 \times \rho \times v^2 \times S \times C_{Do} - \mu \times W} dv$$

$$S_{acc} = S_{acc1} + S_{acc2}$$

In this expression the term T_o and T_{ofail} corresponds to the static thrust levels corresponding respectively to the all operating and "critical engine inoperative" configurations.

Horizontal distance necessary to the rotation and transition

In order to estimate the distance expanded during the rotation and transition the time used to rotate and reach the 35 ft altitude was used. This approach is used by Matingly [25] and Anderson [8]. In order to estimate the time necessary to rotate and climbs, several videos were observed. These videos showed take-offs sequences recorded from the cockpit of A320. The take-off procedure requires the non-flying pilot to announce when the aircraft reaches V1, VR, lifts off and reaches V2 at the 35 ft obstacle clearance position. By measuring the time between the announcement of VR and lift-off, the rotation duration was measured. In a similar fashion, the time necessary to transition was measured using the time difference between VLOF and the obstacle clearance. (Note: These videos were publically available on www.youtube.com and www.FL350.com). By taking the average on the measured time the following duration were estimated:

$\Delta t_R = 2.78$s: Time to rotate (little variation in the measurements)

$\Delta t_{Tr} = 3$s: Time necessary to clear 35ft obstacle after lift off (larger dispersion in the measurements compared to time to rotate)

The distance necessary to these segments were estimated by taking an average between the speeds bracketing each.

(68)

$$S_R = \frac{VR + VFLO}{2} \times \Delta t_R$$
$$S_{Tr} = \frac{VFLO + V2}{2} \times \Delta t_{Tr}$$

The speed targets VR, VFLO and V2 were defined using publically available A320 performance tables.

Defining Thrust requirements from the BLF and STOFL distances

Based on the analysis described above the BLF and STOFL distances were defined:

(69)

$$STOFL = S_{acc}(T_0) + S_R + S_{Tr}$$
$$BFL = S_{acc}(T_0, \delta) + S_R + S_{Tr}$$

The acceleration distance based on the integrals in equation (66) and (67) were solved step-wise using the principle shown in equation (65). In order to solve for the thrust levels that will yield the STOFL and BFL targets respectively, a root-finding algorithm was build based on the bisection method.

A.2.3 Thrust requirement calculator

Based on the analyses presented above, a calculator was developed to estimate the thrust requirements necessary to perform the manoeuvres specified by certification requirements and customer requirements. The calculator was implemented in Excel with visual basic macro. The calculator was organized by spreadsheets.

The mission and vehicle spreadsheet

Table 49: Vehicle description

Vehicle parameters			
TOGM	76911	[kg]	Take off gross mass
TOGW	169560	[lb]	Take off gross weight
	754497	[N]	
Sw	122	[m2]	Wing area
Vehicle performance parameters			
Vehicle conf:		**Gnd**	
Cdo	0.0548	[-]	Profile drag coef.
mu_roll	0.03	[-]	LG Rolling friction coef.
mu_brak	0.2	[-]	LG braking coef.
V1	146	[knot]	Decision speed
VR	152	[knot]	Rotation speed
V2	154	[knot]	Lift off speed
Vehicle conf:		**TOLG**	
Cdi	0.0875	[-]	Induced drag coef.
Cd1	-0.081	[-]	Interference drag coef.
Cdo	0.0548	[-]	Profile drag coef.
Clmax	2.04	[-]	Max lift. coef
Vehicle conf:		**TO**	
Cdi	0.0875	[-]	Induced drag coef.
Cd1	-0.081	[-]	Interference drag coef.
Cdo	0.0498	[-]	Profile drag coef.
Vehicle conf:		**Enroute**	
Tables			
Vehicle conf:		**LD**	
Cdi	0.0979	[-]	Induced drag coef.
Cd1	-0.14	[-]	Interference drag coef.
Cdo	0.2204	[-]	Profile drag coef.
Vehicle conf:		**LDLG**	
Cdi	0.0979	[-]	Induced drag coef.
Cd1	-0.14	[-]	Interference drag coef.
Cdo	0.2254	[-]	Profile drag coef.

This spreadsheet presents the main assumptions used in the translation of the vehicle requirements into thrust requirements. This calculator assumes that the vehicle aerodynamic and weight are fixed. The description of the vehicle was based on the parameters listed in Table 49 (extracted from the calculator).

The mass of used for the Take Off Gross Mass (TOGM) and wing area Sw were based on the value used by the EDS team in their representation of the A320 (for validation purposes).

The aerodynamic parameters listed above are based on the aerodynamic analysis presented earlier. For each flight phase, the aerodynamic coefficients listed in this table are used to represent the relevant wing configuration.

In addition to the vehicle description this spreadsheet provides an overview of key requirements. This description is provided by Table 50. This table lists the gradients required by the certification authorities along with excess power and target field lengths for take off. This table centralizes the parameters which are transmitted to the scenario calculators. This table also provides a description of the flight phases used to defined each scenario.

Table 50: Mission and requirements

Climb Gradients	TOCL1	15	0.5	0	[%]	Climb gradients for FAR/JAR regulated manoeuvres
	TOCL2	7.5	3	2.4		
	TOCLfin	5	1.7	1.2		
	AppCL	5	2.7	2.1		
	AbdLD	3.2	2	0		
Excess power	PCL_acc	1500	700	500	[ft/min]	Excess power for climb transition acceleration
	EnrouteCL	500	400	300	[ft/min]	Excess power for enroute climb
	CzCLpt	250	-	-	[ft/min]	Excess power for final climb (nominal thrust-Phase 6)

Flight phases				**Ground operation requirements**	
Phases	Altitude [FL]	Mach [-]	Description Main phase (symmetric phase)	Short take off field length (ShTOFL):	
				1725	[m]
1	0	0	Park (-)	Balanced take-off field length (BalFL):	
2	0	0	Engine start (-)	2300	[m]
3	0	0.04	Taxi (-)	TOFL ratio:	75%
4	0	0.04	Take-off (Landing)	Acceleration in taxi (Taxi_a):	
5	15	0.35	Initial climb (Approach)	1	[m/s2]
6	275	0.75	Final climb (Initial descent)		
7	350	0.78	Cruise		
8	150	0.71	Initial descent (Final climb)		
9	100	0.54	holding (Climb acceleration)		
10	100	0.45	Approach (Initial climb)		
11	0	0.3	Landing (Take-off initial climb)		
12	100	0.54	Emer. Cruise		

Scenario calculators

The calculation associated with each scenario is centralized in a spreadsheet allowing the user to understand how the mission requirements translated into thrust requirements. These requirements are extracted from the mission input spreadsheet. The general layout of the information is presented in Figure 246. The part in box 1 highlights the calculations characterizing the ambient conditions (pressure, temperature, sonic speed,

etc…). These calculations are based on the altitude and flight Mach number received from the input spreadsheet. For each scenario, several requirements may be enforced. In the scenario presented in Figure 246, three requirements are used to define the thrust requirements. Calculations related to the thrust requirements are preformed in boxes 2.1, 2.2 and 2.3. All the results are collected in box 3.

Some of the numbers are coloured. The colours are used to highlight the fact that these numbers are extracted from the input spreadsheet. The orange numbers refers to weight characteristics. The green represent aerodynamic characteristics. Clear blue refer to mission parameters and purple to requirements. The dark blue colour signals the calculated thrust requirements.

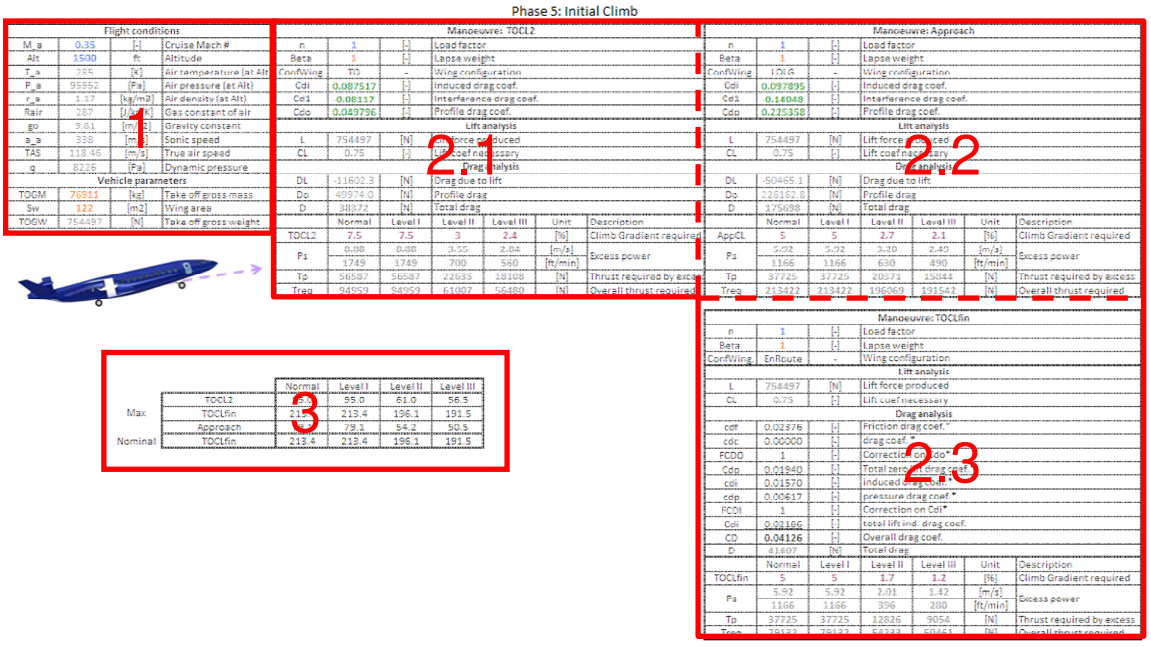


Figure 246: Thrust requirements calculator layout

The thrust requirements mostly consist in climb gradients and excess power. These requirements were translated into thrust requirements using the explicit equations (56) to (59). Therefore for all excess power and climb requirements, the thrust was determined in a linear fashion

The calculator associated with the take-off field length requirement was set up slightly differently. Equation (69) provides an explicit function of the take-off distance as a function of static thrust. The inverse of this function was not derived. Therefore, translating take-off field length requirements into thrust requirements could not be done explicitly and necessitated the implementation of a root finding routine. This calculator is based on a macro which has to be triggered manually using the button above. The calculator is based on a two step optimizer. The first optimization loop determines the static thrust necessary to take-off at the specified STOFL (i.e. the field length requirement when no propulsive subsystems are failed). Then based on this thrust estimation it will estimate the smallest thrust necessary to meet the balanced field length if the propulsive system was to fail at V1. Since the take-off phase is evaluated for static condition, all thrust requirements are adjusted using the lapse function presented earlier. With this adjustment the static thrust necessary to meet the take-off requirement is define.

Validation

The calculator was validated comparing the results obtained by the calculator and those from the EDS/FLOPS results. For the take-off thrust calculation was validated by estimating the thrust required for a standard A320 (no narrow foot print take-off procedure) The thrust required obtained was within 5% of the thrust required by two CFM 56-5 (engines currently used to operate A320).

Thrust requirements for the mission

In the previous paragraph we introduced a series of requirements which apply to different flight phases, at different failure criticality levels. The following discussion explains how these requirements were mapped to the flight phases. Two general rules were applied in associating vehicle to thrust requirements:

- The requirements applied for normal conditions were also applied to scenario of criticality I (dispatchable failures). Generally these requirements are defined based on customer requirements.
- Failures of criticality type II and III were mostly based on requirements defined by certification requirements.

Airport operation phases (Parked at gate, engine start and taxi): Phase 1-3

In phases 1 and 2 (Parked at gate and engine start), no thrust requirements were imposed. In phase 3 (taxi) which is valid only for normal configuration, the maximum thrust required was based on a taxi scenario with friction from the landing. The requirement imposes the aircraft to accelerate on a flat surface at a rate of 1 m/s^2. The nominal thrust required based on the aerodynamic drag and ground resistance. The calculation associated with the evaluation of the thrust requirements are available in the following table (extracted from the calculator).

Table 51: Thrust calculations for ground operation

Flight conditions			
M_a	0.04	[-]	Cruise Mach #
Alt	0	ft	Altitude
T_a	288	[K]	Air temperature (at Alt)
P_a	101325	[Pa]	Air pressure (at Alt)
r_a	1.23	[kg/m3]	Air density (at Alt)
Rair	287	[J/kg/K]	Gas constant of air
go	9.81	[m/s2]	Gravity constant
a_a	340	[m/s]	Sonic speed
TAS	13.61	[m/s]	True air speed
q	113	[Pa]	Dynamic pressure
Vehicle parameters			
TOGM	76911	[kg]	Take off gross mass
Sw	122	[m2]	Wing area
TOGW	754497	[N]	Take off gross weight
Manoeuvre: Taxi			
Taxi_a	1	[m/s2]	Acceleration
Beta	1	[-]	Lapse weight
ConfWing.	Gnd	-	Wing configuration
Cdo	0.0548	[-]	Profile drag coef.
mu	0.03	[-]	Friction coef. from LG
Resistive forces analysis			
Do	758.3	[N]	Profile drag
fr	22635	[N]	Friction from Landing Gears
Ta	76911	[N]	Thrust required by accel.
Treq	100304	[N]	Acceleration thrust
Treq	23393	[N]	Thrust taxi

Take-off phase: Phase 4

To determine the thrust required in the take-off phase, the amount of static thrust necessary to take off within the specified take-off field length distances was determined along with the amount of thrust necessary to perform the take-climb segment 1 gradient (TOCL1). The requirements for the take-off distance were defined as customer requirements and were corresponding to 1725m for regular take-off and 2300m for the failure case (note the failed case is the balanced field length which counts as the official TOFL distance for FAR/JAR specification).

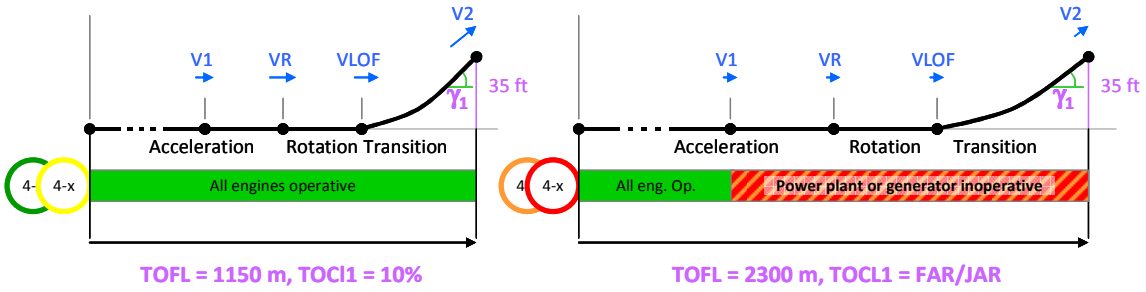


Figure 247: Overview of take off requirements

Table 52: Take off calculation inputs

Airport/ambient air conditions			
Alt	0	ft	Altitude of take off
Beta	1	[-]	Lapse weight
T_a	288	[K]	Air temperature (at Alt)
P_a	101325	[Pa]	Air pressure (at Alt)
r_a	1.23	[kg/m3]	Air density (at Alt)
Rair	287	[J/kg/K]	Gas constant of air
go	9.81	[m/s2]	Gravity constant
a_a	340	[m/s]	Sonic speed
Takeoff certification rules FAR 25.103/107/113			
kFTO	118%	[-]	Final speed coefficient
hObst	35	[ft]	Virtual obstacle height
htr	400	[ft]	Transition altitude
hClimb	1,500	[ft]	Transition to climb phase
Vehicle parameters			
TOGM	76911	[kg]	Take off gross mass
Sw	122	[m2]	Wing area
n	1	[-]	load factor
TOGW	754496.9	[N]	Take off gross weight
Vehicle performances			
Cdo	0.054796	[-]	Profile drag coef.
Clmax	2.04	[-]	Maximum lift coefficient
mu	0.03	[-]	Friction from Landing gears
Maneuvres target			
STOFL	1725	[m]	Short TOFL (normal op.)
BFL	2300	[m]	Balanced field length (FAR req)
V1	146	[knot]	Decision speed
VR	152	[knot]	Rotation speed
V2	154	[knot]	Lift off speed
Approximate time			
t_rot	2.78	[s]	Time necessary to rotate
t_tr	3	[s]	Time necessary for transition

Based on the input presented in

Table 52, the following thrust requirements were determined:

Table 53: Thrust requirement calculation for take off run

Acceleration distances			
s_acc1	1155	[m]	Distance pre V1 (fully op.)
s_acc2	693	[m]	Distance post V1 (failed)
s_accTot	1273	[m]	Total accel dist (fully op.)
Rotation and transition distances			
s_r	217	[m]	rotation distance
s_tr	235	[m]	transition distance
Total distances			
STOFL	**1725**	[m]	Take-off field length (fully op.)
TOFL	**2300**	[m]	Balanced TOFL (FAR rule)
Field length thrust requirements			
Th_STOFL	**298495**	[N]	Thrust required for STOFL target
Th_TOFL	**106488**	[N]	Thrust required for BLF target

The thrust requirements presented above were computed iteratively. The convergence process is presented in the following graph:

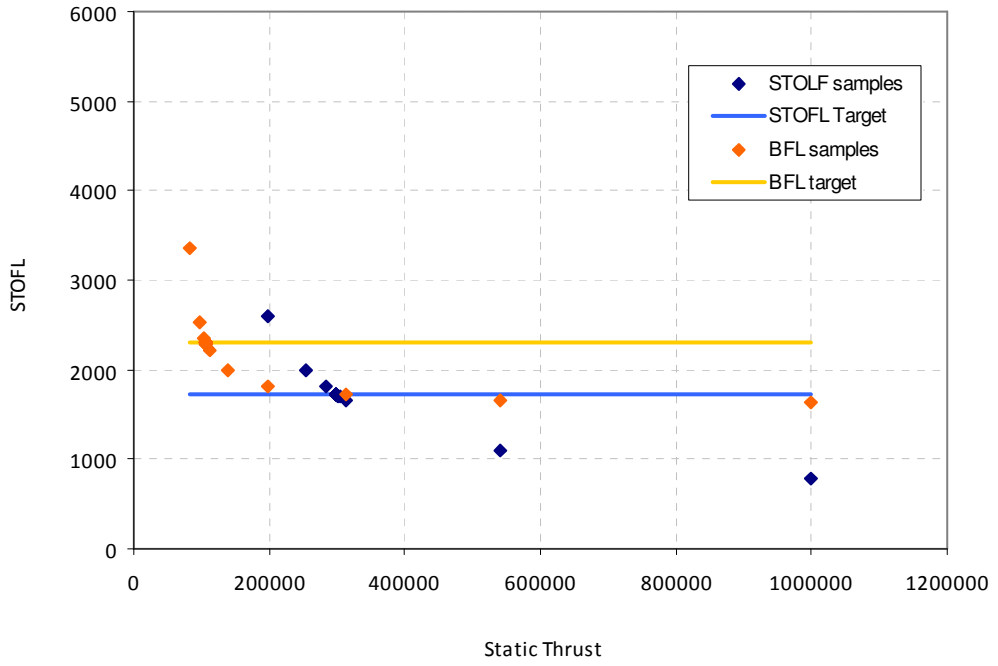


Figure 248: Convergence process toward thrust requirements

In addition to the take-off run distance the first climb segment requirement was unforced (with a thrust lapse adjustment). This thrust requirements was defined so that the aircraft is able to perform the climb gradients specified earlier while accelerating from VLOF to V2 within the time dedicated for transition (3 s). The following table presents the steps in the calculation of this requirement.

Table 54: TOCL1 thrust requirements calculations

Aerodynamic performances			
ConfWing.	TOLG	-	Wing configuration
Cdi	0.08751699	[-]	Induced drag coef.
Cd1	-0.0811737	[-]	Interference drag coef.
Cdo	0.05479611	[-]	Profile drag coef.
Lift analysis			
L	754497	[N]	Lift force produced
q	3745	[Pa]	Dynamic pressure at lift-off
CL	1.65	[-]	Lift coef necessary
Drag analysis			
DL	47787.0	[N]	Drag due to lift
Do	25038.1	[N]	Profile drag
D	72825	[N]	Total drag
Thrust analysis			
M_FLO	0.230	[-]	Mach number at lift-off
Lapse	0.692	[-]	thrust lapse at lift-off
FN	1	[N]	Total thrust

Manoeuvre: TOCL1						
	Normal	Level I	Level II	Level III	Unit	Description
TOCL1	15	15	0.5	0	[%]	Climb Gradient required
Ps	14.46	14.46	3.12	2.73	[m/s]	Excess power
	2847	2847	615	538	[ft/min]	
Tp	139552	139552	30150	26378	[N]	Thrust required by excess power
Tn	212377	212377	102975	99203	[N]	Net thrust required at lift-off
To	306785	306785	148751	143301	[N]	Static thrust required

Initial climb phase: Phase 5

The maximum thrust required during this phase will be determined by estimating the thrust levels necessary to:

- the 2nd segment take-off-climb,
- the final take-off-climb segments and,
- the missed approach (by symmetry in flight conditions).

In this phase, the two requirements will be calculated and the most constraining will be used for the max thrust requirement. The nominal thrust requirement will be calculated based on the final take-off-climb requirement as it will be in this regime that the engine will operate the longest in the initial climb phase. The degradation standards used for both 2nd take-off climb and final take-off-climb segments will be based on the degradation profile derived from certification rules (Table 47). The calculations associated with each of these constraints are provided in the following tables.

Table 55: Phase 5 flight conditions

Flight conditions			
M_a	0.35	[-]	Cruise Mach #
Alt	1500	ft	Altitude
T_a	285	[K]	Air temperature (at Alt)
P_a	95952	[Pa]	Air pressure (at Alt)
r_a	1.17	[kg/m3]	Air density (at Alt)
Rair	287	[J/kg/K]	Gas constant of air
go	9.81	[m/s2]	Gravity constant
a_a	338	[m/s]	Sonic speed
TAS	118.46	[m/s]	True air speed
q	8226	[Pa]	Dynamic pressure

Table 56: Phase 5 maneuver thrust requirements calculations

Manoeuvre: TOCL2						
n	1	[-]	Load factor			
Beta	1	[-]	Lapse weight			
ConfWing.	TO	-	Wing configuration			
Cdi	0.087517	[-]	Induced drag coef.			
Cd1	-0.081174	[-]	Interference drag coef.			
Cdo	0.049796	[-]	Profile drag coef.			
Lift analysis						
L	754497	[N]	Lift force produced			
CL	0.75	[-]	Lift coef necessary			
Drag analysis						
DL	-11602.3	[N]	Drag due to lift			
Do	49974.0	[N]	Profile drag			
D	38372	[N]	Total drag			
	Normal	Level I	Level II	Level III	Unit	Description
TOCL2	7.5	7.5	3	2.4	[%]	Climb Gradient required
Ps	8.88	8.88	3.55	2.84	[m/s]	Excess power
	1749	1749	700	560	[ft/min]	
Tp	56587	56587	22635	18108	[N]	Thrust required by excess
Treq	94959	94959	61007	56480	[N]	Overall thrust required

Manoeuvre: TOCLfin						
n	1	[-]	Load factor			
Beta	1	[-]	Lapse weight			
ConfWing.	EnRoute	-	Wing configuration			
Lift analysis						
L	754497	[N]	Lift force produced			
CL	0.75	[-]	Lift coef necessary			
Drag analysis						
cdf	0.02376	[-]	Friction drag coef.*			
cdc	0.00000	[-]	drag coef. *			
FCDO	1	[-]	Correction on Cdo*			
Cdo	0.01940	[-]	Total zero lift drag coef.*			
cdi	0.01570	[-]	induced drag coef.*			
cdp	0.00617	[-]	pressure drag coef.*			
FCDI	1	[-]	Correction on Cdi*			
Cdi	0.02186	[-]	total lift ind. drag coef.			
CD	**0.04126**	[-]	Overall drag coef.			
D	41407	[N]	Total drag			
	Normal	Level I	Level II	Level III	Unit	Description
TOCLfin	5	5	1.7	1.2	[%]	Climb Gradient required
Ps	5.92	5.92	2.01	1.42	[m/s]	Excess power
	1166	1166	396	280	[ft/min]	
Tp	37725	37725	12826	9054	[N]	Thrust required by excess
Treq	**79132**	**79132**	**54233**	**50461**	[N]	Overall thrust required

Manoeuvre: Approach						
n	1	[-]	Load factor			
Beta	1	[-]	Lapse weight			
ConfWing.	LDLG	-	Wing configuration			
Cdi	**0.097895**	[-]	Induced drag coef.			
Cd1	**-0.140485**	[-]	Interference drag coef.			
Cdo	**0.225358**	[-]	Profile drag coef.			
Lift analysis						
L	754497	[N]	Lift force produced			
CL	0.75	[-]	Lift coef necessary			
Drag analysis						
DL	-50465.1	[N]	Drag due to lift			
Do	226162.8	[N]	Profile drag			
D	175698	[N]	Total drag			
	Normal	Level I	Level II	Level III	Unit	Description
AppCL	5	5	2.7	2.1	[%]	Climb Gradient required
Ps	5.92	5.92	3.20	2.49	[m/s]	Excess power
	1166	1166	630	490	[ft/min]	
Tp	37725	37725	20371	15844	[N]	Thrust required by excess
Treq	**213422**	**213422**	**196069**	**191542**	[N]	Overall thrust required

Final climb to cruise: Phase 6

Unlike previous phases the thrust requirements will be different between normal/dispatchable failure configurations and deeply critical failure configurations (level II and III). In normal and level I configurations the aircraft must climb to cruise altitude. Therefore in these two states the aircraft must meet the Enroute-climb requirement. For deeper failures, the aircraft is de-routed to an alternate airport and will abort its climb to cruise altitude. The flight phase "ferry to alternate airport" is set to occur at FL 100 which is already below the level corresponding to the final climb to cruise altitude point (FL 275). Therefore if a level II or III failure was to occur, there is no longer a need to climb. As a result the manoeuvre requirement for these failures is only to be able to maintain a level continuous flight attitude. In normal and level I failures, the nominal amount of thrust for the phase will be based on the thrust necessary to a continuous climb at 250 ft/min.

Table 57: Phase 6 flight conditions

Flight conditions			
M_a	0.75	[-]	Cruise Mach #
Alt	27500	ft	Altitude
T_a	234	[K]	Air temperature (at Alt)
P_a	33676	[Pa]	Air pressure (at Alt)
r_a	0.50	[kg/m3]	Air density (at Alt)
Rair	287	[J/kg/K]	Gas constant of air
go	9.81	[m/s2]	Gravity constant
a_a	306	[m/s]	Sonic speed
TAS	229.77	[m/s]	True air speed
q	13251	[Pa]	Dynamic pressure

Table 58: Phase 6 maneuver thrust requirements calculations

Manoeuvre: Enroute Climb/LCF						
n	1.5	[-]	Load factor			
Beta	1	[-]	Lapse weight			
ConfWing.	EnRoute	-	Wing configuration			
Lift analysis						
L	1131745	[N]	Lift force produced			
CL	0.70	[-]	Lift coef necessary			
Drag analysis						
cdf	0.02032	[-]	Friction drag coef.*			
cdc	0.00113	[-]	drag coef. *			
FCDO	1	[-]	Correction on Cdo*			
Cdo	0.01989	[-]	Total zero lift drag coef.*			
cdi	0.01360	[-]	induced drag coef.*			
cdp	0.00493	[-]	pressure drag coef.*			
FCDI	1	[-]	Correction on Cdi*			
Cdi	0.01854	[-]	total lift ind. drag coef.			
CD	**0.03843**	[-]	Overall drag coef.			
D	62118	[N]	Total drag			
	Normal	Level I	Level II	Level III	Unit	Description
ERCL/LCF	500	500	0	0	[ft/min]	Climb Gradient required
Ps	2.54	2.54	0.00	0.00	[m/s]	Excess power
Tp	8341	8341	0	0	[N]	Thrust required by excess
Treq	**70459**	**70459**	**62118**	**62118**	[N]	Overall thrust required

Manoeuvre: Enroute Climb				
n	1	[-]	Load factor	
Beta	1	[-]	Lapse weight	
ConfWing.	EnRoute	-	Wing configuration	
Lift analysis				
L	754497	[N]	Lift force produced	
CL	0.47	[-]	Lift coef necessary	
Drag analysis				
cdf	0.02032	[-]	Friction drag coef.*	
cdc	0.00113	[-]	drag coef. *	
FCDO	1	[-]	Correction on Cdo*	
Cdo	0.01634	[-]	Total zero lift drag coef.*	
cdi	0.00606	[-]	induced drag coef.*	
cdp	0.00116	[-]	pressure drag coef.*	
FCDI	1	[-]	Correction on Cdi*	
Cdi	0.00722	[-]	total lift ind. drag coef.	
CD	**0.02356**	[-]	Overall drag coef.	
D	38089	[N]	Total drag	
	Normal	Level I	Unit	Description
ERCL	250	250	[ft/min]	Climb Gradient required
Ps	1.27	1.27	[m/s]	Excess power
Tp	4170	4170	[N]	Thrust required by excess
Treq	**42260**	**42260**	[N]	Overall thrust required

Cruise: Phase 7

The cruise phase is applicable only for normal and level I failures. The maximum thrust requirement is defined based on the Enroute-climb requirement (enforcing the capability perform the top of climb requirement). The nominal thrust requirement was defined assuming Levelled Continuous Flight conditions.

Table 59: Phase 7 flight conditions

Flight conditions			
M_a	0.78	[-]	Cruise Mach #
Alt	35000	ft	Altitude
T_a	219	[K]	Air temperature (at Alt)
P_a	23842	[Pa]	Air pressure (at Alt)
r_a	0.38	[kg/m3]	Air density (at Alt)
Rair	287	[J/kg/K]	Gas constant of air
go	9.81	[m/s2]	Gravity constant
a_a	296	[m/s]	Sonic speed
TAS	231.24	[m/s]	True air speed
q	10155	[Pa]	Dynamic pressure

Table 60: Phase 7 manoeuvre thrust requirements calculations

Manoeuvre: Enroute climb				
n	1.3	[-]	Load factor	
Beta	1	[-]	Lapse weight	
ConfWing.	EnRoute	-	Wing configuration	
Lift analysis				
L	980846	[N]	Lift force produced	
CL	0.79	[-]	Lift coef necessary	
Drag analysis				
cdf	0.02013	[-]	Friction drag coef.*	
cdc	0.00152	[-]	drag coef. *	
FCDO	1	[-]	Correction on Cdo*	
Cdo	0.02083	[-]	Total zero lift drag coef.*	
cdi	0.01741	[-]	induced drag coef.*	
cdp	0.00960	[-]	pressure drag coef.*	
FCDI	1	[-]	Correction on Cdi*	
Cdi	0.02701	[-]	total lift ind. drag coef.	
CD	**0.04784**	[-]	Overall drag coef.	
D	59268	[N]	Total drag	
	Normal	Level I	Unit	Description
ERCL	500	500	[ft/min]	Climb Gradient required
Ps	2.54	2.54	[m/s]	Excess power
Tp	8288	8288	[N]	Thrust required by excess
Treq	67556	67556	[N]	Overall thrust required

Manoeuvre: Leveled-continuous flight				
n	1	[-]	Load factor	
Beta	1	[-]	Lapse weight	
ConfWing.	EnRoute	-	Wing configuration	
Lift analysis				
L	754497	[N]	Lift force produced	
CL	0.61	[-]	Lift coef necessary	
Drag analysis				
cdf	0.02981	[-]	Friction drag coef.*	
cdc	0.00000	[-]	drag coef. *	
FCDO	1	[-]	Correction on Cdo*	
Cdo	0.02470	[-]	Total zero lift drag coef.*	
cdi	0.01031	[-]	induced drag coef.*	
cdp	0.00313	[-]	pressure drag coef.*	
FCDI	1	[-]	Correction on Cdi*	
Cdi	0.01344	[-]	total lift ind. drag coef.	
CD	**0.03814**	[-]	Overall drag coef.	
D	47248	[N]	Total drag	
	Normal	Level I	Unit	Description
LCF	0	0	[ft/min]	Climb Gradient required
Ps	0.00	0.00	[m/s]	Excess power
Tp	0	0	[N]	Thrust required by excess
Treq	**47248**	**47248**	[N]	Overall thrust required

Initial descent: Phase 8

In initial descent phase the mean thrust requirements are neglected (assumed to be zero). The initial descent is used (by symmetry in its operating conditions) to represent a climb phase. Therefore the maximum thrust requirement will be defined by the Enroute-climb for the normal and levelled continuous flight for flight levels II and III.

Table 61: Phase 8 flight conditions

Flight conditions			
M_a	0.71	[-]	Cruise Mach #
Alt	15000	ft	Altitude
T_a	258	[K]	Air temperature (at Alt)
P_a	57182	[Pa]	Air pressure (at Alt)
r_a	0.77	[kg/m3]	Air density (at Alt)
Rair	287	[J/kg/K]	Gas constant of air
go	9.81	[m/s2]	Gravity constant
a_a	322	[m/s]	Sonic speed
TAS	228.75	[m/s]	True air speed
q	20172	[Pa]	Dynamic pressure

Table 62: Phase 8 maneuver thrust requirements calculations

Manoeuvre: Enroute Climb/LCF						
n	1.5	[-]	Load factor			
Beta	1	[-]	Lapse weight			
ConfWing.	EnRoute	-	Wing configuration			
Lift analysis						
L	1131745	[N]	Lift force produced			
CL	0.46	[-]	Lift coef necessary			
Drag analysis						
cdf	0.02058	[-]	Friction drag coef.*			
cdc	0.00083	[-]	drag coef. *			
FCDO	1	[-]	Correction on Cdo*			
Cdo	0.01874	[-]	Total zero lift drag coef.*			
cdi	0.00588	[-]	induced drag coef.*			
cdp	0.00114	[-]	pressure drag coef.*			
FCDI	1	[-]	Correction on Cdi*			
Cdi	0.00702	[-]	total lift ind. drag coef.			
CD	**0.02576**	[-]	Overall drag coef.			
D	63396	[N]	Total drag			
	Normal	Level I	Level II	Level III	Unit	Description
ERCL/LCF	500	500	0	0	[ft/min]	Climb Gradient required
Ps	2.54	2.54	0.00	0.00	[m/s]	Excess power
Tp	8378	8378	0	0	[N]	Thrust required by excess
Treq	**71773**	**71773**	**63396**	**63396**	[N]	Overall thrust required

Holding and acceleration: Phase 9

This flight phase is applicable only for normal and level I failure (otherwise the failure is considered an emergency which will give priority for landing). This phase, occurring at FL 100 and M=0.6, is identical to the one corresponding to a hypothetical mid-climb acceleration. Therefore the maximum thrust required from this flight phase will be defined by the thrust necessary to accelerate the aircraft with an excess power of 1500 ft/min. The mean thrust requirement is defined base on the thrust necessary to maintain levelled continuous thrust in a clean wing configuration.

Table 63: Phase 9 flight conditions

Flight conditions			
M_a	0.54	[-]	Cruise Mach #
Alt	10000	ft	Altitude
T_a	268	[K]	Air temperature (at Alt)
P_a	69682	[Pa]	Air pressure (at Alt)
r_a	0.90	[kg/m3]	Air density (at Alt)
Rair	287	[J/kg/K]	Gas constant of air
go	9.81	[m/s2]	Gravity constant
a_a	328	[m/s]	Sonic speed
TAS	177.28	[m/s]	True air speed
q	14214	[Pa]	Dynamic pressure

Table 64: Phase 9 maneuver thrust requirements calculations

Manoeuvre: Acceleration				
n	1.3	[-]	Load factor	
Beta	1	[-]	Lapse weight	
ConfWing.	EnRoute	-	Wing configuration	
Ps	1500	[ft/min]	Acceleration	
Lift analysis				
L	980846	[N]	Lift force produced	
CL	0.57	[-]	Lift coef necessary	
Drag analysis				
cdf	0.0219	[-]	Friction drag coef.*	
cdc	0.00006	[-]	drag coef. *	
FCDO	1	[-]	Correction on Cdo*	
Cdo	0.0186	[-]	Total zero lift drag coef.*	
cdi	0.0089	[-]	induced drag coef.*	
cdp	0.0023	[-]	pressure drag coef.*	
FCDI	1	[-]	Correction on Cdi*	
Cdi	0.0112	[-]	total lift ind. drag coef.	
CD	**0.0299**	[-]	Overall drag coef.	
D	51762	[N]	Total drag	
	Normal	Level I	Unit	Description
Ps_acc	7.62	7.62	[m/s]	Excess power
Tp	32430	32430	[N]	Thrust required by excess
Treq	**84192**	**84192**	[N]	Overall thrust required

Manoeuvre: Leveled-continuous flight				
n	1	[-]	Load factor	
Beta	1	[-]	Lapse weight	
ConfWing.	EnRoute	-	Wing configuration	
Lift analysis				
L	754497	[N]	Lift force produced	
CL	0.44	[-]	Lift coef necessary	
Drag analysis				
cdf	0.0219	[-]	Friction drag coef.*	
cdc	0.00006	[-]	drag coef. *	
FCDO	1	[-]	Correction on Cdo*	
Cdo	0.0184	[-]	Total zero lift drag coef.*	
cdi	0.0053	[-]	induced drag coef.*	
cdp	0.0010	[-]	pressure drag coef.*	
FCDI	1	[-]	Correction on Cdi*	
Cdi	0.0062	[-]	total lift ind. drag coef.	
CD	**0.0246**	[-]	Overall drag coef.	
D	42693	[N]	Total drag	
	Normal	Level I	Unit	Description
LCF	0	0	[ft/min]	Climb Gradient required
Ps	0.00	0.00	[m/s]	Excess power
Tp	0	0	[N]	Thrust required by excess
Treq	**42693**	**42693**	[N]	Overall thrust required

Final approach: Phase 10

This flight phase which occurs at FL 10 and M = 0.45 represents the early phase of the approach. The maximum thrust required during this phase will be determined by estimating the thrust levels necessary to:

- the missed approach requirements
- the mid-climb acceleration after the clean to dirty acceleration (by symmetry in the flight conditions) and
- the final take-off-climb segments (by symmetry).

The mean thrust is ignored.

Table 65: Phase 10 flight conditions

Flight conditions			
M_a	0.45	[-]	Cruise Mach #
Alt	10000	ft	Altitude
T_a	268	[K]	Air temperature (at Alt)
P_a	69682	[Pa]	Air pressure (at Alt)
r_a	0.90	[kg/m3]	Air density (at Alt)
Rair	287	[J/kg/K]	Gas constant of air
go	9.81	[m/s2]	Gravity constant
a_a	328	[m/s]	Sonic speed
TAS	147.73	[m/s]	True air speed
q	9871	[Pa]	Dynamic pressure

Table 66: Phase 10 maneuver thrust requirements calculations

Manoeuvre: Approach						
n	1	[-]	Load factor			
Beta	1	[-]	Lapse weight			
ConfWing.	TO	-	Wing configuration			
Cdi	0.087517	[-]	Induced drag coef.			
Cd1	-0.081174	[-]	Interference drag coef.			
Cdo	0.049796	[-]	Profile drag coef.			
Lift analysis						
L	754497	[N]	Lift force produced			
CL	0.63	[-]	Lift coef necessary			
Drag analysis						
DL	-19873.2	[N]	Drag due to lift			
Do	59964.6	[N]	Profile drag			
D	40091	[N]	Total drag			
	Normal	Level I	Level II	Level III	Unit	Description
AppCL	5	5	2.7	2.1	[%]	Climb Gradient required
Ps	7.39	7.39	3.99	3.10	[m/s]	Excess power
	1454	1454	785	611	[ft/min]	
Tp	37725	37725	20371	15844	[N]	Thrust required by excess
Treq	77816	77816	60463	55936	[N]	Overall thrust required

Manoeuvre: Acceleration						
n	1	[-]	Load factor			
Beta	1	[-]	Lapse weight			
ConfWing.	EnRoute	-	Wing configuration			
Lift analysis						
L	754497	[N]	Lift force produced			
CL	0.63	[-]	Lift coef necessary			
Drag analysis						
cdf	0.02266	[-]	Friction drag coef.*			
cdc	0.00000	[-]	drag coef. *			
FCDO	1	[-]	Correction on Cdo*			
Cdo	0.01923	[-]	Total zero lift drag coef.*			
cdi	0.01092	[-]	induced drag coef.*			
cdp	0.00345	[-]	pressure drag coef.*			
FCDI	1	[-]	Correction on Cdi*			
Cdi	0.01437	[-]	total lift ind. drag coef.			
CD	**0.03359**	[-]	Overall drag coef.			
D	40452	[N]	Total drag			
	Normal	Level I	Level II	Level III	Unit	Description
Ps_acc	1500	1500	700	500	[ft/min]	Excess power
	7.62	7.62	3.56	2.54	[m/s]	
Tp	38916	38916	18161	12972	[N]	Thrust required by excess
Treq	**40370**	**40370**	**19615**	**14426**	[N]	Overall thrust required

Manoeuvre: TOCLfin						
n	1	[-]	Load factor			
Beta	1	[-]	Lapse weight			
ConfWing.	EnRoute	-	Wing configuration			
Lift analysis						
L	754497	[N]	Lift force produced			
CL	0.63	[-]	Lift coef necessary			
Drag analysis						
cdf	0.02266	[-]	Friction drag coef.*			
cdc	0.00000	[-]	drag coef. *			
FCDO	1	[-]	Correction on Cdo*			
Cdo	0.01923	[-]	Total zero lift drag coef.*			
cdi	0.01092	[-]	induced drag coef.*			
cdp	0.00345	[-]	pressure drag coef.*			
FCDI	1	[-]	Correction on Cdi*			
Cdi	0.01437	[-]	total lift ind. drag coef.			
CD	**0.03359**	[-]	Overall drag coef.			
D	40452	[N]	Total drag			
	Normal	Level I	Level II	Level III	Unit	Description
TOCLfin	5	5	1.7	1.2	[%]	Climb Gradient required
Ps	7.39	7.39	2.51	1.77	[m/s]	Excess power
	1454	1454	494	349	[ft/min]	
Tp	37725	37725	12826	9054	[N]	Thrust required by excess
Treq	**78177**	**78177**	**53278**	**49506**	[N]	Overall thrust required

Landing/Aborted Landing: Phase 11

The maximum thrust required in this phase is driven by the following requirements:

- Aborted landing climb requirement
- the 1st segment take-off-climb (by symmetry in flight conditions) and,
- the 2nd segment take-off-climb (by symmetry).

Nominal thrust is ignored.

Table 67: Phase 11 flight conditions

Flight conditions			
M_a	0.3	[-]	Cruise Mach #
Alt	0	ft	Altitude
T_a	288	[K]	Air temperature (at Alt)
P_a	101325	[Pa]	Air pressure (at Alt)
r_a	1.23	[kg/m3]	Air density (at Alt)
Rair	287	[J/kg/K]	Gas constant of air
go	9.81	[m/s2]	Gravity constant
a_a	340	[m/s]	Sonic speed
TAS	102.06	[m/s]	True air speed
q	6380	[Pa]	Dynamic pressure

Table 68: Phase 11 maneuver thrust requirements calculations

Manoeuvre: Aborted landing						
n	1	[-]	Load factor			
Beta	1	[-]	Lapse weight			
ConfWing.	LD+Gear	-	Wing configuration			
Cdi	0.0979	[-]	Induced drag coef.			
Cd1	-0.1405	[-]	Interference drag coef.			
Cdo	0.2254	[-]	Profile drag coef.			
Lift analysis						
L	754497	[N]	Lift force produced			
CL	0.97	[-]	Lift coef necessary			
Drag analysis						
DL	-34402.0	[N]	Drag due to lift			
Do	38761.5	[N]	Profile drag			
D	4360	[N]	Total drag			
	Normal	Level I	Level II	Level III	Unit	Description
AbdLD	3.2	3.2	2	0	[%]	Climb Gradient required
Ps	3.27	3.27	2.04	0.00	[m/s]	Excess power
	643	643	402	0	[ft/min]	
Tp	24144	24144	15090	0	[N]	Thrust required by excess
Treq	165161	165161	156107	141017	[N]	Overall thrust required

Manoeuvre: TOCL1						
n	1	[-]	Load factor			
Beta	1	[-]	Lapse weight			
ConfWing.	LDLG	-	Wing configuration			
Cdi	0.097895	[-]	Induced drag coef.			
Cd1	-0.140485	[-]	Interference drag coef.			
Cdo	0.225358	[-]	Profile drag coef.			
Lift analysis						
L	754497	[N]	Lift force produced			
CL	0.97	[-]	Lift coef necessary			
Drag analysis						
DL	-34402.0	[N]	Drag due to lift			
Do	175419.4	[N]	Profile drag			
D	141017	[N]	Total drag			
	Normal	Level I	Level II	Level III	Unit	Description
TOCL1	15	15	0.5	0	[%]	Climb Gradient required
Ps	15.31	15.31	0.51	0.00	[m/s]	Excess power
	3014	3014	100	0	[ft/min]	
Tp	113175	113175	3772	0	[N]	Thrust required by excess
Treq	254192	254192	144790	141017	[N]	Overall thrust required

Manoeuvre: TOCL2						
n	1	[-]	Load factor			
Beta	1	[-]	Lapse weight			
ConfWing.	TO	-	Wing configuration			
Cdi	0.087517	[-]	Induced drag coef.			
Cd1	-0.081174	[-]	Interference drag coef.			
Cdo	0.049796	[-]	Profile drag coef.			
Lift analysis						
L	754497	[N]	Lift force produced			
CL	0.97	[-]	Lift coef necessary			
Drag analysis						
DL	2757.9	[N]	Drag due to lift			
Do	38761.5	[N]	Profile drag			
D	41519	[N]	Total drag			
	Normal	Level I	Level II	Level III	Unit	Description
TOCL2	7.5	7.5	3	2.4	[%]	Climb Gradient required
Ps	7.65	7.65	3.06	2.45	[m/s]	Excess power
	1507	1507	603	482	[ft/min]	
Tp	56587	56587	22635	18108	[N]	Thrust required by excess
Treq	98107	98107	64154	59627	[N]	Overall thrust required

Emergency ferry: Phase 12

This phase is defined only for level II and III failures. The maximum thrust is defined for a degraded enroute-climb requirement and the mean thrust is defined by the thrust necessary to sustain levelled continuous flight conditions.

Table 69: Phase 12 flight conditions

Flight conditions			
M_a	0.54	[-]	Cruise Mach #
Alt	10000	ft	Altitude
T_a	268	[K]	Air temperature (at Alt)
P_a	69682	[Pa]	Air pressure (at Alt)
r_a	0.90	[kg/m3]	Air density (at Alt)
Rair	287	[J/kg/K]	Gas constant of air
go	9.81	[m/s2]	Gravity constant
a_a	328	[m/s]	Sonic speed
TAS	177.28	[m/s]	True air speed
q	14214	[Pa]	Dynamic pressure

Table 70: Phase 12 maneuver thrust requirements calculations

Manoeuvre: Acceleration				
n	1.3	[-]	Load factor	
Beta	1	[-]	Lapse weight	
ConfWing.	EnRoute	-	Wing configuration	
Lift analysis				
L	980846	[N]	Lift force produced	
CL	0.57	[-]	Lift coef necessary	
Drag analysis				
cdf	0.0219	[-]	Friction drag coef.*	
cdc	0.00006	[-]	drag coef. *	
FCDO	1	[-]	Correction on Cdo*	
Cdo	0.0186	[-]	Total zero lift drag coef.*	
cdi	0.0089	[-]	induced drag coef.*	
cdp	0.0023	[-]	pressure drag coef.*	
FCDI	1	[-]	Correction on Cdi*	
Cdi	0.0112	[-]	total lift ind. drag coef.	
CD	**0.0299**	[-]	Overall drag coef.	
D	51762	[N]	Total drag	
	Level II	Level III	Unit	Description
Ps_acc	400	300	[ft/s]	Excess power
	2	2	[m/s]	
Tp	8648	6486	[N]	Thrust required by excess
Treq	**60410**	**58248**	[N]	Overall thrust required

Manoeuvre: Leveled-continuous flight				
n	1	[-]	Load factor	
Beta	1	[-]	Lapse weight	
ConfWing.	EnRoute	-	Wing configuration	
Lift analysis				
L	754497	[N]	Lift force produced	
CL	0.44	[-]	Lift coef necessary	
Drag analysis				
cdf	0.0219	[-]	Friction drag coef.*	
cdc	0.00006	[-]	drag coef. *	
FCDO	1	[-]	Correction on Cdo*	
Cdo	0.0184	[-]	Total zero lift drag coef.*	
cdi	0.0053	[-]	induced drag coef.*	
cdp	0.0010	[-]	pressure drag coef.*	
FCDI	1	[-]	Correction on Cdi*	
Cdi	0.0062	[-]	total lift ind. drag coef.	
CD	**0.0246**	[-]	Overall drag coef.	
D	42693	[N]	Total drag	
	Level II	Level III	Unit	Description
LCF	0	0	[ft/min]	Climb Gradient required
Ps	0.00	0.00	[m/s]	Excess power
Tp	0	0	[N]	Thrust required by excess
Treq	**42693**	**42693**	[N]	Overall thrust required

Overview of requirements

The following tables provide an overview of the assignment of the various vehicle performance requirements to the flight phases and criticality levels considered in the study. Table 71 and

Table 72 present the thrust levels associated with each requires. This table allows us to see which requirement is active in each phase/criticality level.

Table 71: Req. for max thrust

#	Phase reference	FL	MN	Requirements types				Requirements in kN of thrust			
				Nominal	Level I	Level II	Level III	Nominal	Level I	Level II	Level III
1	Park	0	0					0			
2	Eng. Start	0	0					0			
3	Taxi	0	0.04	Taxi				100.3			
4	Take-off	0	0.04	TOLF	TOLF	TOLF	TOLF	298.5	298.5	106.5	106.5
				TOCl1	TOCl1	TOCl1	TOCl1	306.8	306.8	148.8	143.3
5	Initial clim	15	0.35	TOCl2	TOCl2	TOCl2	TOCl2	95.0	95.0	61.0	56.5
				TOClFin	TOClFin	TOClFin	TOClFin	213.4	213.4	196.1	191.5
				Approach	Approach	Approach	Approach	79.1	79.1	54.2	50.5
6	Final climb	275	0.75	EnrouteCl	EnrouteCl	LCF	LCF	70.5	70.5	62.1	62.1
7	Cruise	350	0.78	EnrouteCl	EnrouteCl			67.6	67.6		
8	Initial desc	150	0.71	EnrouteCl	EnrouteCl	LCF	LCF	71.8	71.8	63.4	63.4
9	Holding	100	0.54	Accel	Accel			84.2	84.2		
10	Final appro	100	0.45	TOClFin	TOClFin	TOClFin	TOClFin	77.8	77.8	60.5	55.9
				Accel	Accel	Accel	Accel	40.4	40.4	19.6	14.4
				Approach	Approach	Approach	Approach	78.2	78.2	53.3	49.5
11	Landing	0	0.3	TOCl1	TOCl1	TOCl1	TOCl1	254.2	254.2	144.8	141.0
				TOCl2	TOCl2	TOCl2	TOCl2	98.1	98.1	64.2	59.6
				AbdLD	AbdLD	AbdLD	AbdLD	165.2	165.2	156.1	141.0
12	Emergency	100	0.54			EnrouteCl	EnrouteCl			60	58

Table 72: Req. for nominal thrust

#	Phase reference	FL	MN	Requirements types				Requirements in kN of thrust			
				Nominal	Level I	Level II	Level III	Nominal	Level I	Level II	Level III
1	Park	0	0					0			
2	Eng. Start	0	0					0			
3	Taxi	0	0.04	LCF				23.4			
4	Take-off	0	0	TOLF	TOLF	TOLF	TOLF	298.5	298.5	106.5	106.5
5	Initial clim	0	0	TOClFin	TOClFin	TOClFin	TOClFin	213.4	213.4	196.1	191.5
6	Final climb	0	0	LCF	LCF	LCF	LCF	42.3	42.3	0	0
7	Cruise	0	0	LCF	LCF			47.2	47.2		
8	Initial desc	0	0					0	0	0	0
9	Holding	0	0	LCF	LCF			42.7	42.7		
10	Final apprc	0	0					0	0	0	0
11	Landing	0	0					0	0	0	0
12	Emergency	0	0			LCF	LCF			43	43

The following tables represent the thrust requirement summary for all scenarios. This format of table will be used to provide an overview for each functional requirement presented in the following sections.

Table 73: Functional requirements for thrust force

		Ground				Taxi		Take-Off		Climb			
	Criticality level	1		2		3		4		5		6	
		Max	Av	Max	Av	Max	Av	Max	Av	Max	Av	Max	Av
Thrust	Normal	0	0	0	0	100	23	307	298	213	213	70	42
	Level I							307	298	213	213	70	42
	Level II							149	106	196	196	62	0
	Level III							143	106	192	192	62	0
	Units	[kN]											

		Cruise		Holding		Descent				Land	
	Criticality level	7/12		9		8		10		11	
		Max	Av	Max	Av	Max	Av	Max	Av	Max	Av
Thrust	Normal	68	47	84	43	72	0	78	0	254	0
	Level I	68	47	84	43	72	0	78	0	254	0
	Level II	60	43			63	0	60	0	156	0
	Level III	58	43			63	0	56	0	141	0
	Units	[kN]									

A.3 Customer and technical loads

A.3.1 Functional breakdown

This functional group includes two functions. The fist one is the customer energy functionality. This function refers to energy requirements dedicated to hosting passengers. These loads will be organized on the following categories:

- In-Flight Entertainment systems (IFE)
- Galleys
- Cabin lighting

The other function is referred to energy dedicated to miscellaneous technical functions. This function is assumed to include things like avionics, maintenance (health monitoring), communication and safety systems and aircraft external lighting. Note this definition of technical load does not include the load implied by the environment conditioning system or wing anti-icing system.

A.3.2 Quantification of the maximum functional requirements

A.3.3 Customer loads

The functionalities associated with customer are organized as follows:

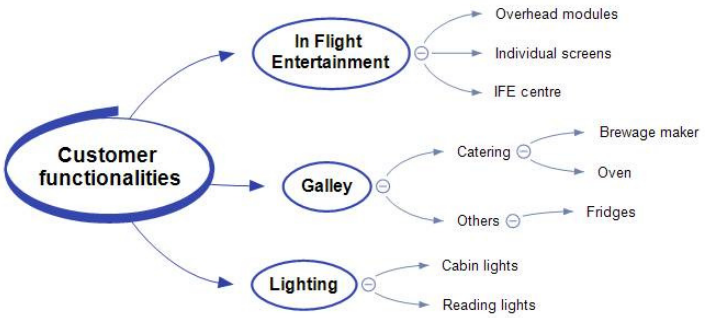


Figure 249: Customer functionalities breakdown

The requirements associated were defined using EADS models.

Normal requirement profile

Some functionalities will be operating in a continuous fashion (their power is constant over the normal mission). These functions include the fridges, the avionics and cabin safety equipments will operate at continuously during the mission. Their maximum and average power loads are assumed to be 100% and 75% respectively of the loads prescribed in Figure 249 The difference in maximum and average loads assumes that some of their components are intermittently switched on and off. For the other sub-functions we need to consider more closely what is happening in each phase.

During flight phase 1, it is assumed that everything can be performed (or tested simultaneously). Hence this flight phase requires maximum loads for each sub-function to be provided.

In the engine start phase (phase 2), the functional requirements are decreased to ease the sizing constraints on the energy source (the batteries). In this phase, the IFE power is limited to the IFE-centre (the overheads and individual screens are off). The catering systems are prevented from operating. Based on the model used, the IFE-centre represents 15% of the overall power accounted for the IFE sub-function. During engine start the cabin lights are also deactivated but overhead light can be activated individually by passenger. Based on the model used, the overhead lights represent 45% of the total load assigned to IFE. Assuming that on average only half the passenger will switch their overhead lights the average power load was assumed to be 23% for lights. Finally, the aircraft health monitoring functionality is momentarily placed on standby mode (or we can assume that only the engine is monitored).

In the taxi out phase, the IFE functionality is partially reactivated with overhead screens which increase the load to 30%. Also catering functionalities are temporarily reactivated (they will be deactivated again during take off and initial climb). The health monitoring functionality is fully active and will remain that state through the rest of the

normal mission. The catering and IFE functionalities are fully activated between the final climb and initial descent phases. The average IFE load listed (86% of max) assumes that 80% of passengers are using their individual screens which overall account for 70% of the total IFE load.

Based on the observation and assumptions described above, following functional profile will be required from the power architecture:

Table 74: Functional profile in normal operation

			Ground				Taxi		TakeOff		Climb				Cruise		Loiter		Descent				Landing	
			1		2		3		4		5		6		7		9		8		10		11	
			Max.	Av.	Max.	Av.	Max.	Av.	Max.	Av.	Max.	Av.	Max.	Av.	Max.	Av.	Max.	Av.	Max.	Av.	Max.	Av.	Max.	Av.
Cust. Loads	IFE	Max of normal op. [kW]	100%	100%	15%	15%	30%	30%	30%	30%	30%	30%	100%	86%	100%	86%	100%	86%	100%	86%	30%	30%	30%	30%
	Catering (ovens)		100%	100%	0%	0%	100%	50%	0%	0%	0%	0%	100%	50%	100%	50%	100%	50%	100%	50%	0%	0%	0%	0%
	Fridges		100%	100%	100%	75%	100%	75%	100%	75%	100%	75%	100%	75%	100%	75%	100%	75%	100%	75%	100%	75%	100%	75%
	Lights		100%	100%	45%	23%	45%	23%	45%	23%	45%	23%	100%	64%	100%	55%	100%	55%	100%	64%	45%	23%	45%	23%
Tech. Loads	Avionics		100%	100%	100%	75%	100%	75%	100%	75%	100%	75%	100%	75%	100%	75%	100%	75%	100%	75%	100%	75%	100%	75%
	Cabin safety eq.		100%	100%	100%	75%	100%	75%	100%	75%	100%	75%	100%	75%	100%	75%	100%	75%	100%	75%	100%	75%	100%	75%
	Health mon.		100%	100%	20%	15%	100%	75%	100%	75%	100%	75%	100%	75%	100%	75%	100%	75%	100%	75%	100%	75%	100%	75%
	Lighting		100%	100%	50%	50%	100%	100%	100%	40%	100%	25%	100%	5%	100%	5%	100%	5%	100%	5%	100%	25%	100%	40%

Level I degradation profile

In a level I failure, the mission must be continued with little effect on passenger comfort and no degradation of the overall safety of the aircraft. Therefore none of the technical loads are degraded. Among the customer loads, the IFE is degraded with the deactivation of individual screens which account for 70% of the load. Nevertheless, the IFE functionality is maintained by the continued operation of the overhead screens and the IFE-centre. The catering functionality is also slightly degraded by a limitation on the number of ovens that can be active at once. Based on these degradations, the functional profile presented in Table 75 was defined.

Table 75: Degradation profile I for the customer and technical loads

			TakeOff		Climb				Cruise		Loiter		Descent				Landing	
			4		5		6		7		9		8		10		11	
			Max.	Av.	Max.	Av.	Max.	Av.	Max.	Av.	Max.	Av.	Max.	Av.	Max.	Av.	Max.	Av.
Cust. Loads	IFE	Max of normal op. [kW]	30%	30%	30%	30%	30%	30%	30%	30%	30%	30%	0%	30%	0%	0%	0%	0%
	Catering (ovens)		0%	0%	0%	0%	75%	38%	75%	38%	0%	0%	0%	0%	0%	0%	0%	0%
	Refrigeration		100%	75%	100%	75%	100%	75%	100%	75%	100%	75%	100%	75%	100%	75%	100%	75%
	Cabin lighting		45%	23%	45%	23%	100%	64%	100%	55%	100%	55%	100%	64%	45%	23%	45%	23%
Tech. Loads	Avionics		100%	75%	100%	75%	100%	75%	100%	75%	100%	75%	100%	75%	100%	75%	100%	75%
	Cabin safety eq.		100%	75%	100%	75%	100%	75%	100%	75%	100%	75%	100%	75%	100%	75%	100%	75%
	Health mon.		100%	75%	100%	75%	100%	75%	100%	75%	100%	75%	100%	75%	100%	75%	100%	75%
	Lighting		100%	40%	100%	25%	100%	5%	100%	5%	100%	5%	100%	5%	100%	25%	100%	40%

Level II degradation profile

If a failure of criticality II occurs, functionalities dedicated to passenger comforts are put on hold. This shedding concerns the IFE, catering (including fridges) functionalities. The emergency cabin lights are activated. The health monitoring which is dedicated to maintenance activities is also eliminated (the most vital subsystems continue to be monitored by the subsystems classified under avionics). The aircraft external lighting functionalities are reduced to a "bare minimum".

Table 76: Degradation profile II for the customer and technical loads

			TakeOff		Climb				Cruise		Loiter		Descent				Landing	
			4		5		6		7		9		8		10		11	
			Max.	Av.	Max.	Av.	Max.	Av.	Max.	Av.	Max.	Av.	Max.	Av.	Max.	Av.	Max.	Av.
Cust. Loads	IFE	Max of normal op. [kW]	0%	0%	0%	0%	0%	0%	0%	0%			0%	0%	0%	0%	0%	0%
	Catering (ovens)		0%	0%	0%	0%	0%	0%	0%	0%			0%	0%	0%	0%	0%	0%
	Refrigeration		0%	0%	0%	0%	0%	0%	0%	0%			0%	0%	0%	0%	0%	0%
	Cabin lighting		45%	0%	0%	0%	45%	23%	45%	23%			45%	23%	100%	55%	100%	55%
Tech. Loads	Avionics		100%	0%	0%	0%	0%	0%	100%	75%			100%	75%	100%	75%	100%	75%
	Cabin safety eq.		0%	0%	0%	0%	0%	0%	100%	75%			100%	75%	100%	75%	100%	75%
	Health mon.		0%	0%	0%	0%	0%	0%	0%	0%			0%	0%	0%	0%	0%	0%
	Lighting		35%	25%	35%	25%	20%	5%	20%	5%			20%	5%	35%	25%	35%	25%

Level III degradation profile

In this dire situation, only avionics are maintained (in a minimum power configuration) along with the cabin and external lighting functionalities in their emergency configurations.

Table 77: Degradation profile III for the customer and technical loads

			TakeOff		Climb				Cruise		Loiter		Descent				Landing	
			4		5		6		7		9		8		10		11	
			Max.	Av.	Max.	Av.	Max.	Av.	Max.	Av.	Max.	Av.	Max.	Av.	Max.	Av.	Max.	Av.
Cust. Loads	IFE	Max of normal op. [kW]	0%	0%	0%	0%	0%	0%	0%	0%			0%	0%	0%	0%	0%	0%
	Catering (ovens)		0%	0%	0%	0%	0%	0%	0%	0%			0%	0%	0%	0%	0%	0%
	Refrigeration		0%	0%	0%	0%	0%	0%	0%	0%			0%	0%	0%	0%	0%	0%
	Cabin lighting		10%	5%	10%	5%	10%	5%	10%	5%			0%	0%	0%	0%	0%	0%
Tech. Loads	Avionics		51%	38%	51%	38%	51%	38%	51%	38%			0%	38%	0%	38%	51%	38%
	Cabin safety eq.		0%	0%	0%	0%	0%	0%	0%	0%			0%	0%	0%	0%	0%	0%
	Health mon.		0%	0%	0%	0%	0%	0%	0%	0%			0%	0%	0%	0%	0%	0%
	Lighting		35%	25%	35%	25%	20%	5%	20%	5%			20%	5%	35%	25%	35%	25%

Overview of requirements

The requirements from the customer and technical energy functions are modelled as three functional requirements:

- Fixed frequency alternating current (FFAC) customer loads, including the IFE, refrigeration and cabin lighting
- Variable frequency alternating current (VFAC) customer loads, including catering (ovens)
- Fixed frequency alternating current (FFAC) technical loads

A.4 Wing icing protection

The icing protection function can operate in 2 modes: anti-icing and de-icing. In anti-icing mode, the wing leading edge is heated in fashion which will prevent icing accretion to form on the surface. This operation requires the surface to be continuously heated. In de-icing mode, the leading edge is either physically deformed of heated intermittently in order to force aggregation of ice to detach from the leading edge. The de-icing mode requires less energy than anti-icing mode.

In this experiment, the loads will represented by the requirements imposed by electric mats. These mats are resistors transforming electrical energy into heat. These mats can operate in the two modes described earlier by either being on continuously (anti-icing mode) or intermittently to provide the heat flashes necessary to de-icing. In both mode of operation the anti-icing subsystem will require VFAC power. Their operation is also assumed to require continuous energy (ie. The maximum and average power loads are assumed to be the same).

A.4.1 Quantification of the maximum functional requirements

The model used to estimate the amount of power necessary to protect the wings from accretion of ice was defined by Susan Liscouet-Hanke. A description of this model can be found in her thesis [22].

A.4.2 Overview of requirements

To derive the degraded profiles the WIP was considered as a safety related functionality which priority decreases with altitude (highly important at low altitude, less important at higher altitudes). Based on this assumption, anti-icing is required during taxi, take off, landing and low altitude climb/descent. On the ground, anti-icing mode must be available with the exception of the engine start (unnecessary load which would oversize the battery). In high altitude conditions (final climb, cruise or early descent) anti-icing capability is not strictly necessary de-icing modes are made available in the high altitude climb and descent phases.

Since WIP is considered as a safety related function, the level I profiles failures do not allow for degradation. For deeper criticality levels, the requirements are degraded from anti-icing to de-icing mode with the elimination of the requirements at higher altitude phases.

A.5 Controls

This functionality refers to the actuation of elements influencing directly or indirectly the attitude of the aircraft in flight and on the ground. This function is organized in two main sub-functions: primary and secondary controls.

A.5.1 Functional breakdown

The primary flight control functionality refers to the actuation of control surfaces which can directly change the attitude of the aircraft in flight. These controlled surfaces include the ailerons, spoilers, rudder and elevator. The control of these surfaces is essential to the survival of the aircraft and will never be shed or degraded is their requirements. The secondary flight control refers to mechanical actuations dedicated to the deployment of high lift devices (flaps and slats), trimming the aircraft in pitch (horizontal tail surfaces), and actuation of the landing gear deployment, retraction, steering and braking. The primary and secondary controls sub-functionalities will be organized as follows:

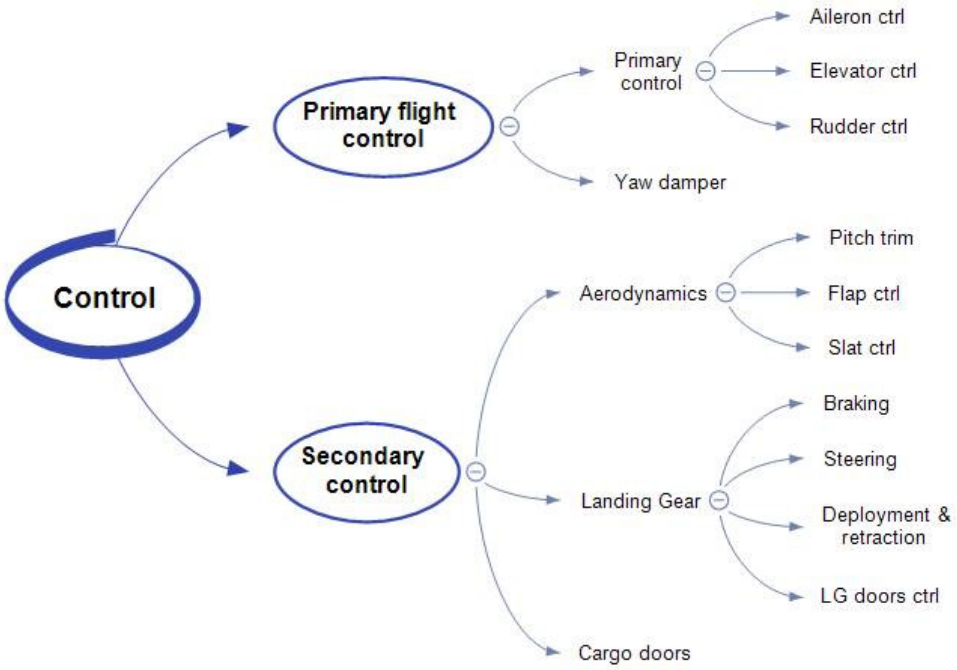


Figure 250: Control function breakdown

A.5.2 Process used in determination of functional requirements

The estimation of the power loads induced by these functions was derived from reference[88]. This document includes a description of the hydraulic flows required in each flight phase for an A320. The loads listed in this graph were registered in a table and assigned to the various sub-functions composing the control functionality. The graph is presented in Figure 251, and the assignments are presented in Table 79. Based on the information gathered in Table 79, the normal load levels per flight phase were identified. These loads were transformed into electric requirement from the hydraulic flows based on the hydraulic power equation. These levels are presented in the following table per sub-functions:

Table 78: Normal load levels for control

	Ground				Taxi		Take-Off		Climb			
	1		2		3		4		5		6	
	Max	Av	Max	Av	Max	Av	Max	Av	Max	Av	Max	Av
Prim	**3.62**	0	**0**	0	**3.62**	0.36	**14.8**	11.1	**14.8**	11.1	**22.3**	18.8
Yaw damper	**0**	0	**0**	0	**0**	0	**0**	0	**0**	0	**53.7**	0
Pitch trim	**10**	0	**0**	0	**10**	1	**10**	1	**10**	0.08	**12.7**	0.13
Slats	**7.44**	0	**0**	0	**7.44**	0.74	**7.44**	0.62	**11.2**	0.07	**0**	0
Flaps	**6.2**	0	**0**	0	**6.2**	0.62	**6.2**	0.62	**18.2**	0.36	**4.53**	0.09
Brakes	**0**	0	**0**	0	**9.83**	0.98	**0**	0	**0**	0	**0**	0
LG dep.ret	**0**	0	**0**	0	**0**	0	**0**	0	**18.8**	0.38	**0**	0
LG Doors	**0**	0	**0**	0	**0**	0	**0**	0	**16.1**	0.29	**0**	0
Steering	**0**	0	**0**	0	**4.48**	0.45	**0**	0	**0**	0	**0**	0
Cargo Doors	**4.36**	0	**0**	0	**0**	0	**0**	0	**0**	0	**0**	0

DC Loads in [kW]

	Cruise		Holding		Descent				Land	
	7		9		8		10		11	
	Max	Av	Max	Av	Max	Av	Max	Av	Max	Av
Prim	**22.3**	18.8	**22.3**	18.8	**22.3**	18.8	**17.2**	8.35	**13.7**	6.83
Yaw damper	**53.7**	0	**53.7**	0	**53.7**	0	**0**	0	**0**	0
Pitch trim	**12.7**	0.13	**12.7**	0.21	**12.7**	0.21	**17.8**	0.36	**0**	0
Slats	**0**	0	**0**	0	**0**	0	**12.9**	0.27	**0**	0
Flaps	**0**	0	**14.1**	0.28	**0**	0	**14.1**	0.28	**0**	0
Brakes	**0**	0	**0**	0	**0**	0	**0**	0	**19.7**	9.83
LG dep.ret	**0**	0	**0**	0	**0**	0	**9.23**	0.18	**0**	0
LG Doors	**0**	0	**0**	0	**0**	0	**11.4**	0.26	**0**	0
Steering	**0**	0	**0**	0	**0**	0	**0**	0	**0**	0
Cargo Doors	**0**	0	**0**	0	**0**	0	**0**	0	**0**	0

DC Loads in [kW]

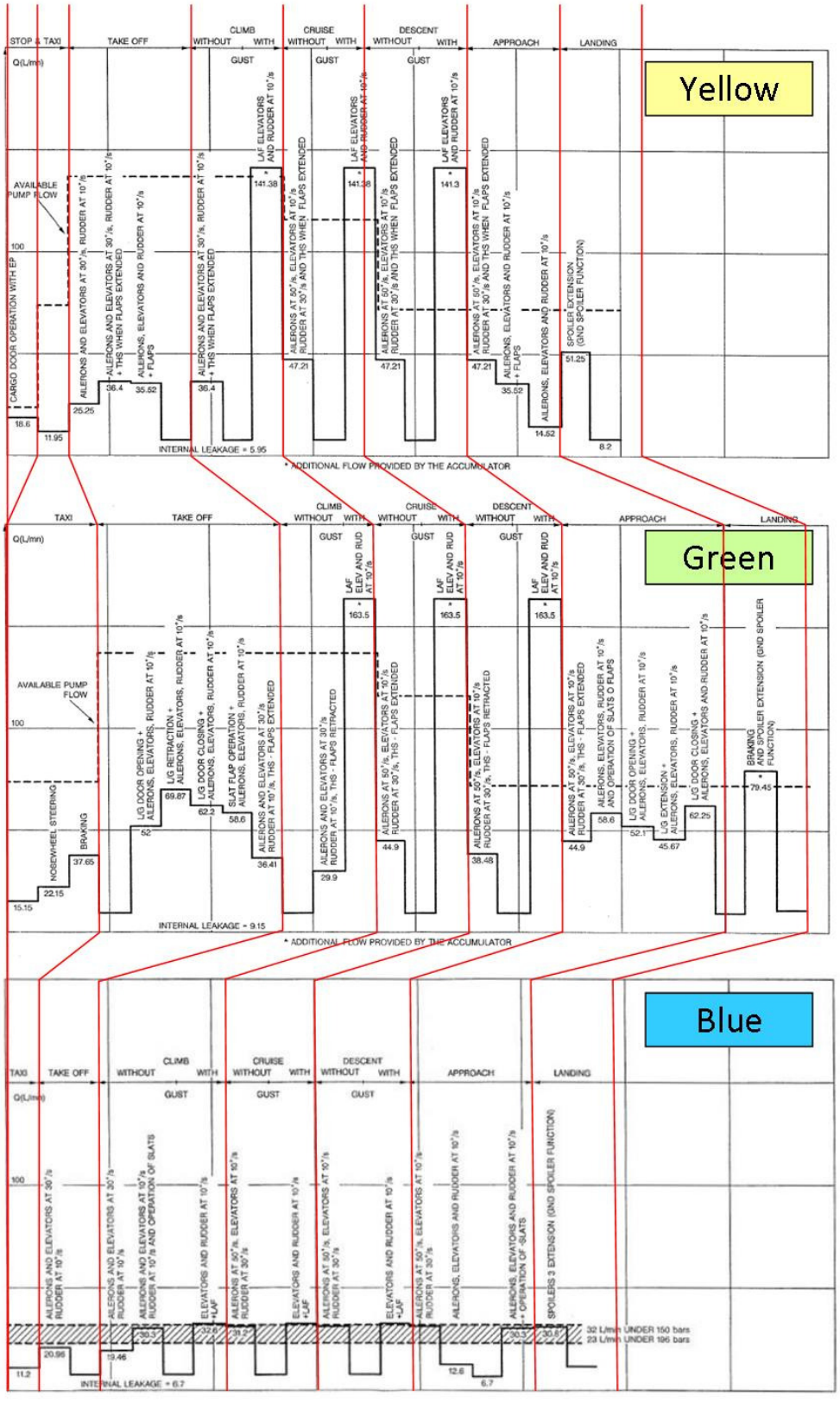


Figure 251: Hydraulic load profile for an A320[89]

Table 79: Flows assigned to the control sub-functions

		Stop	Taxi	take off				climb			cruise			Descent			Approach			Landing		Leak.
	Blue		11.2	21	6.7			19.5	30.3	6.7	31.2	6.7	31.2	31.2	6.7	31.2	12.6	6.7	30.3	30.9	6.7	6.7
	Green		15.2	22.2	37.7	9.15	52	69.9	62.2	59.6	36.4	9.15	29.9	18.4	44.9	9.15	18.4	44.9	59.6	52.1	45.7	62.3	79.5	9.15	9.15
	Yellow	19.6	12	25.3	39.4	35.5	5.95					36.4	5.95	14.1	47.2	5.95	14.1	47.2	35.5	14.5			51.3	6.2	5.95
Blue	Prim		4.5	14.3								12.9			24.5			24.5	5.9		5.9		24.1		
	Slat											10.9	12.9								17.7				
	Flap										13.1		10.9												
Green	Prim				6.27	6.27	6.27	6.27	9.27						20.6			20.6							
	Yaw damper								19				20.6	15.6											
	Pitch trim													13.9											
	Slats							21.6							15			8.58							
	Flaps							21.6																	
	Brakes			26.5																					
	Landing Gear					54.5																			
	LG Doors				36.6																				
	Steering						46.8																		
			13																						
yellow	Prim	12.7	6	19.3	19.3	11.6						19.3			19.3			19.3							
	Yaw damper												14.5	14.5											
	Pitch trim			11.2								11.2		12.1	22			22							
	Flap					19																			
	Cargo Doors																								
	Brakes																								
Assigned flow (net of leakages) [L/min]																									

A.5.3 Normal and degradation profile

Normal and Level I requirement profiles

Table 80 : Normal control requirement profile

		Ground		Taxi	TO	Climb		Cruise	Loiter	Descent		LD
		1	2	3	4	5	6	7	9	8	10	11
Prim	Normal/Level I operation	100%	0%	100%	100%	100%	100%	100%	100%	100%	100%	100%
Yaw damper		0%	0%	0%	0%	0%	100%	100%	100%	100%	0%	0%
Pitch trim		100%	0%	100%	100%	100%	100%	100%	100%	100%	100%	0%
Slats		100%	0%	100%	100%	100%	0%	0%	0%	0%	100%	0%
Flaps		100%	0%	100%	100%	100%	100%	0%	100%	100%	100%	0%
Brakes		0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	100%
LG dep.ret		0%	0%	0%	0%	100%	0%	0%	0%	0%	100%	0%
LG Doors		0%	0%	0%	0%	100%	0%	0%	0%	0%	100%	0%
Steering		0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%
Cargo Doors		100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%

This table provides a summarized overview of when each function occurs in the mission.

Level II degradation profile

In a level II failure configuration, the mission is aborted and the aircraft will attempt to land at the closest suitable airfield. In this situation, the landing gear will remain extended if the failure occurs during take off or initial climb phases (in other words the landing gear retraction process is aborted). Also we shall assume that the requirements specifying the deployment speed of the flaps, slats and the trimming function can be degraded. This degradation results in a 50% decrease in load associated with these functions. The availability of trimming the aircraft in cruise is also eliminated. Also the yaw damper functionality is shed.

Table 81: Level II control requirement profile

		TO	Climb		Cruise	Loiter	Descent		LD
		4	5	6	7	9	8	10	11
Prim	Level II failure	100%	100%	100%	100%		100%	100%	100%
Yaw damper		0%	0%	0%	0%		0%	0%	0%
Pitch trim		50%	50%	50%	0%		50%	50%	50%
Slats		50%	50%	0%	0%		0%	50%	0%
Flaps		50%	50%	50%	0%		50%	50%	0%
Brakes		0%	0%	0%	0%		0%	0%	100%
LG dep.ret		0%	0%	0%	0%		0%	50%	0%
LG Doors		0%	0%	0%	0%		0%	50%	0%
Steering		0%	0%	0%	0%		0%	0%	0%
Cargo Doors		0%	0%	0%	0%		0%	0%	0%

Level III degradation profile

In this deeply critical failure configuration, all secondary control functionalities are shed with the exception with the flap, slat and trim adjustments during the approach. These adjustments are indeed critical to survival of the aircraft as no landing can be attempted without deploying setting the wings their high lift configuration. In a similar fashion, the landing gear is expected to be "dropped" (i.e. deployed under the effects of gravity and aerodynamic forces). For obvious reasons, the breaking functionality during landing can not be eliminated or degraded.

Table 82: Level III control requirement profile

		TO	Climb		Cruise	Loiter	Descent		LD
		4	5	6	7	9	8	10	11
Prim	Level III failure	100%	100%	100%	100%		100%	100%	100%
Yaw damper		0%	0%	0%	0%		0%	0%	0%
Pitch trim		0%	0%	0%	0%		0%	50%	0%
Slats		0%	0%	0%	0%		0%	50%	0%
Flaps		0%	0%	0%	0%		0%	50%	0%
Brakes		0%	0%	0%	0%		0%	0%	100%
LG dep.ret		0%	0%	0%	0%		0%	0%	0%
LG Doors		0%	0%	0%	0%		0%	0%	0%
Steering		0%	0%	0%	0%		0%	0%	0%
Cargo Doors		0%	0%	0%	0%		0%	0%	0%

A.5.4 Overview of requirements

Based on the degradation profiles specified above the following power requirements will be imposed on the architecture.

Table 83: Load required by the primary control function

		Ground				Taxi		Take-Off		Climb			
	Criticality	1		2		3		4		5		6	
	level	Max	Av	Max	Av	Max	Av	Max	Av	Max	Av	Max	Av
Prim. Ctrl. Loads	Normal	4	0	0	0	4	0	15	11	15	11	76	19
	Level I							15	11	15	11	76	19
	Level II							15	11	15	11	22	19
	Level III							15	11	15	11	22	19
	Units	DC [kW]											

		Cruise		Holding		Descent				Land	
	Criticality	7/12		9		8		10		11	
	level	Max	Av	Max	Av	Max	Av	Max	Av	Max	Av
Prim. Ctrl. Loads	Normal	76	19	76	19	76	19	17	8	14	7
	Level I	76	19	76	19	76	19	17	8	14	7
	Level II	22	19			22	19	17	8	14	7
	Level III	22	19			22	19	17	8	14	7
	Units	DC [kW]									

Table 84: Loads required by the secondary control function

		Ground				Taxi		Take-Off		Climb			
	Criticality	1		2		3		4		5		6	
	level	Max	Av	Max	Av	Max	Av	Max	Av	Max	Av	Max	Av
Sec. Ctrl. Load	Normal	28	0	0	0	38	4	24	2	74	1	17	0
	Level I							24	2	74	1	17	0
	Level II							12	1	20	0	9	0
	Level III							0	0	0	0	0	0
	Units	DC [kW]											

		Cruise		Holding		Descent				Land	
	Criticality	7/12		9		8		10		11	
	level	Max	Av	Max	Av	Max	Av	Max	Av	Max	Av
Sec. Ctrl. Load	Normal	13	0	27	0	13	0	65	1	20	10
	Level I	13	0	27	0	13	0	65	1	20	10
	Level II	0	0			6	0	33	1	20	10
	Level III	0	0			0	0	22	0	20	10
	Units	DC [kW]									

A.6 Environment conditioning

A.6.1 Functional breakdown

In this context the environment conditioning functionality refers to the ventilation, pressurization and temperature control of the cabin. In order to fine tune the degradation profiles, this functionality was broken down into three sub-functions:

- Provide power for controlling main cabin environment (including the cargo compartments)
- Provide power for controlling the flight deck environment (including avionic bay)
- Provide power for the emergency oxygen system

A.6.2 Process used in determination of functional requirements

The requirement values were based on data provided by Airbus S.A.S.

A.6.3 Degradation profile

Note the degradation profiles proposed in this section are not based on any study. The effects and consequences of the degradations are only based on educated guesses.

Normal requirement profile

Table 85: Environment conditioning normal operation

		Ground		Taxi	TakeOff	Climb		Cruise	Loiter	Descent		Landing
Cabin/cargo	Max of normal op. [kW]	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
Flightdeck		100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
Oxy-masks		0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%

The degradation profiles were determined as a function of normal operation requirement. Hence Table 85 shows that the requirements associated with the cabin and flight deck are fully performed. On the other hand, the oxygen mask (i.e. emergency oxygen) functionality is not required in the normal mission profile.

Level I degradation profile

Table 86: Environment conditioning level I operation

		Degradation as a function of normal operation							
		TakeOff	Climb		Cruise	Loiter	Descent		Landing
Cabin/cargo	Max of normal op. [kW]	90%	90%	90%	90%	90%	90%	90%	90%
Flightdeck		100%	100%	100%	100%	100%	100%	100%	100%
Oxy-masks		0%	0%	0%	0%	0%	0%	0%	0%

In this degradation profile only the cabin functionality is slightly affected. This degradation is expected to result in a slightly lower pressure in the cabin and increase in temperature. This degradation is not expected to cause major discomfort to passengers. Degradation of flight deck ventilation being considered as a safety-related functionality is not degraded.

Level II failure

Table 87: Environment conditioning level II operation

		Degradation as a function of normal operation							
		TakeOff	Climb		Cruise	Loiter	Descent		Landing
Cabin/cargo	Max of normal op. [kW]	80%	80%	80%	80%		80%	80%	80%
Flightdeck		100%	100%	100%	100%		100%	100%	100%
Oxy-masks		0%	0%	0%	0%		0%	0%	0%

The criticality associated with level II failure allows for a significant degradation of passenger comfort. This degradation is expected to result in a significant lower pressure in the cabin and increase in temperature which could cause passenger discomfort if performed during a long period of time. No degradation of flight deck ventilation is tolerated.

Level III failure

Table 88: Environment conditioning level II operation

		Degradation as a function of normal operation							
		TakeOff	Climb		Cruise	Loiter	Descent		Landing
Cabin/cargo	Max of normal op. [kW]	0%	0%	0%	0%		0%	0%	0%
Flightdeck		0%	0%	0%	0%		0%	0%	0%
Oxy-masks		100%	100%	100%	100%		100%	100%	100%

At this criticality level the elimination of centralized environment conditioning functionalities is tolerated as all effort should be dedicated to functionality strictly related to the recovery or survival of the aircraft. Therefore, neither the cabin nor the flight decks are expected to be pressurized or cooled. The basic needs of both passenger and crew are provided by the emergency functionality.

A.6.4 Implementation of the Requirements

The Matlab code implementing the *Mission* bloc is presented below.

```
% variable: Scenario string[] input
% variable: Criticality string[] input
% variable: FlightPhase string[] input
% variable: ReqMECS double[] output
% variable: ReqnomECS double[] output
% variable: ReqMPrimCtrl double[] output
% variable: ReqnomPrimCtrl double[] output
% variable: ReqMSecCtrl double[] output
% variable: ReqnomSecCtrl double[] output
% variable: ReqMWIP double[] output
% variable: ReqnomWIP double[] output
% variable: ReqMCustFF double[] output
% variable: ReqnomCustFF double[] output
% variable: ReqMCustVF double[] output
% variable: ReqnomCustVF double[] output
% variable: ReqMTechFF double[] output
% variable: ReqnomTechFF double[] output
% variable: Max_Thrust double[] output
% variable: Nom_Thrust double[] output
% variable: MN_Thrust double[] output
% variable: Alt_Thrust double[] output
% variable: Cruisek double output
% variable: DT double[] output
%% Loading information about the mission
%[ECSTable,PrimCtrlTable,SecCtrlTable,WIPTable,CustFFTable,CustVFTable,TechFFTable,Thrust
Table,MNTable,AltTable]=BFTablesEADS();
[ECSTable,PrimCtrlTable,SecCtrlTable,WIPTable,CustFFTable,CustVFTable,TechFFTable,ThrustT
able,MNTable,AltTable]=BFTablesGT();
Durations=DurationTablesShortRange();

FailureLevels=['0_normal','1_dispatch','2_major','3_Hazardous'];
Phases                                                                          =
['Park&Maintenance','EngineStart','Taxi','TakeOff','Climb1','Climb2','Cruise','InitialDes
cent','Holding','FinalApproach','Land','EnRouteEmergency'];

K=length(Criticality); %total number of states
NPhases=length(Phases);
NFailureLevels=length(FailureLevels);

%% Initialization of requirements variables and evaluation of durations:
ReqMECS=zeros(K,1);ReqnomECS=zeros(K,1);
ReqMPrimCtrl=zeros(K,1);ReqnomPrimCtrl=zeros(K,1);
ReqMSecCtrl=zeros(K,1);ReqnomSecCtrl=zeros(K,1);
ReqMWIP=zeros(K,1);ReqnomWIP=zeros(K,1);
ReqMCustFF=zeros(K,1);ReqnomCustFF=zeros(K,1);
ReqMCustVF=zeros(K,1);ReqnomCustVF=zeros(K,1);
ReqMTechFF=zeros(K,1);ReqnomTechFF=zeros(K,1);
Max_Thrust=zeros(K,1);Nom_Thrust=zeros(K,1);
MN_Thrust=zeros(K,1);Alt_Thrust=zeros(K,1);
DT = zeros(K,1);

for k=1:K
    np=0;% phase scanner
    while np<NPhases
        np=np+1;
        if strcmp(Phases[np],FlightPhase[k])
            phase=np;
```

```
            np=NPhases+1;
        end
        if np==NPhases+1
            fl=0;%Failure level scanner
            while fl<NFailureLevels
                fl=fl+1;
                if strcmp(FailureLevels[fl],Criticality[k])
                    critLevel=fl;
                    fl=NFailureLevels;
                end
            end
            CriticalityNb[1,k]=Criticality[k];CriticalityNb[2,k]=critLevel;
            PhaseNb[1,k]=FlightPhase[k];PhaseNb[2,k]=phase;
            % Getting the duration of the scenario
            DT(k) = Durations(critLevel,phase);

            % Getting the correct requirement
            ReqMECS(k)=ECSTable(1,critLevel,phase);
            ReqnomECS(k)=ECSTable(2,critLevel,phase);
            ReqMPrimCtrl(k)=PrimCtrlTable(1,critLevel,phase);
            ReqnomPrimCtrl(k)=PrimCtrlTable(2,critLevel,phase);
            ReqMSecCtrl(k)=SecCtrlTable(1,critLevel,phase);
            ReqnomSecCtrl(k)=SecCtrlTable(2,critLevel,phase);
            ReqMWIP(k)=WIPTable(1,critLevel,phase);
            ReqnomWIP(k)=WIPTable(2,critLevel,phase);
            ReqMCustFF(k)=CustFFTable(1,critLevel,phase);
            ReqnomCustFF(k)=CustFFTable(2,critLevel,phase);
            ReqMCustVF(k)=CustVFTable(1,critLevel,phase);
            ReqnomCustVF(k)=CustVFTable(2,critLevel,phase);
            ReqMTechFF(k)=TechFFTable(1,critLevel,phase);
            ReqnomTechFF(k)=TechFFTable(2,critLevel,phase);
            Max_Thrust(k)=ThrustTable(1,critLevel,phase);
            Nom_Thrust(k)=ThrustTable(2,critLevel,phase);
            MN_Thrust(k)=MNTable(phase);
            Alt_Thrust(k)=AltTable(phase)*100;
        else
            Null=0;DT(k)= -1;
            ReqMECS(k)=Null;        ReqnomECS(k)=Null;
            ReqMPrimCtrl(k)=Null;   ReqnomPrimCtrl(k)=Null;
            ReqMSecCtrl(k)=Null;       ReqnomSecCtrl(k)=Null;
            ReqMWIP(k)=Null;           ReqnomWIP(k)=Null;
            ReqMCustFF(k)=Null;        ReqnomCustFF(k)=Null;
            ReqMCustVF(k)=Null;        ReqnomCustVF(k)=Null;
            ReqMTechFF(k)=Null;        ReqnomTechFF(k)=Null;
            Max_Thrust(k)=Null;        Nom_Thrust(k)=Null;
            MN_Thrust(k)=Null;       Alt_Thrust(k)=Null;
        end
    end
end

%% Fixing the units
Max_Thrust=1000*Max_Thrust;Nom_Thrust=1000*Nom_Thrust;%From kN to N


%% Localisation of cruise point
Cruisek = 0;k = 0;
while Cruisek==0 && k<K
    k=k+1;
    if strcmp(FlightPhase(k),'Cruise')
        Cruisek=k;
        k=K+10;
    end
end
if k==K
    Cruisek=min(K,5);
end


%% Definition of power characteristics
Vnom_DC = zeros(K,1)+10000;
Vnom_VFAC = zeros(K,1)+240;
Vnom_FFAC = zeros(K,1)+300;

SpRat=zeros(K,1)+1.2;
RPMnom=zeros(K,1)+2000;
```

Appendix B

Electric Fan performance and sizing model

B.1 General description of the system

The electric duct fan engine is composed of three main elements: the duct, the fan and the electric motor. The role of the duct is to condition the flow, before it can be processed by the fan, and to expend it in a way that will maximize the thrust generated by the engine. The fan will energize the flow using the mechanical energy provided by the electric motor. A graphical overview of the system is provided in the following figure:

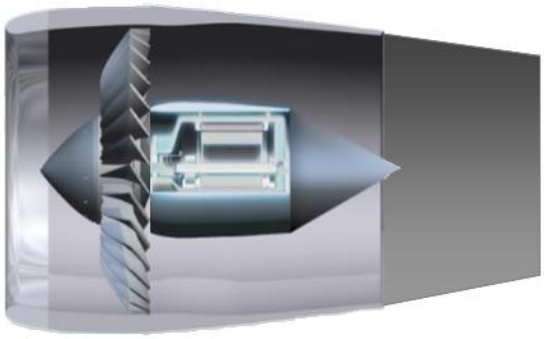


Figure 252: Notional representation of the ducted electric fan[77]

B.2 General description of the sizing model

The sizing model used in this work will be composed of two main elements:

- An optimizer which provides design variables
- A performance model defining the capacity and attributes of the engines under the prescribed mission conditions

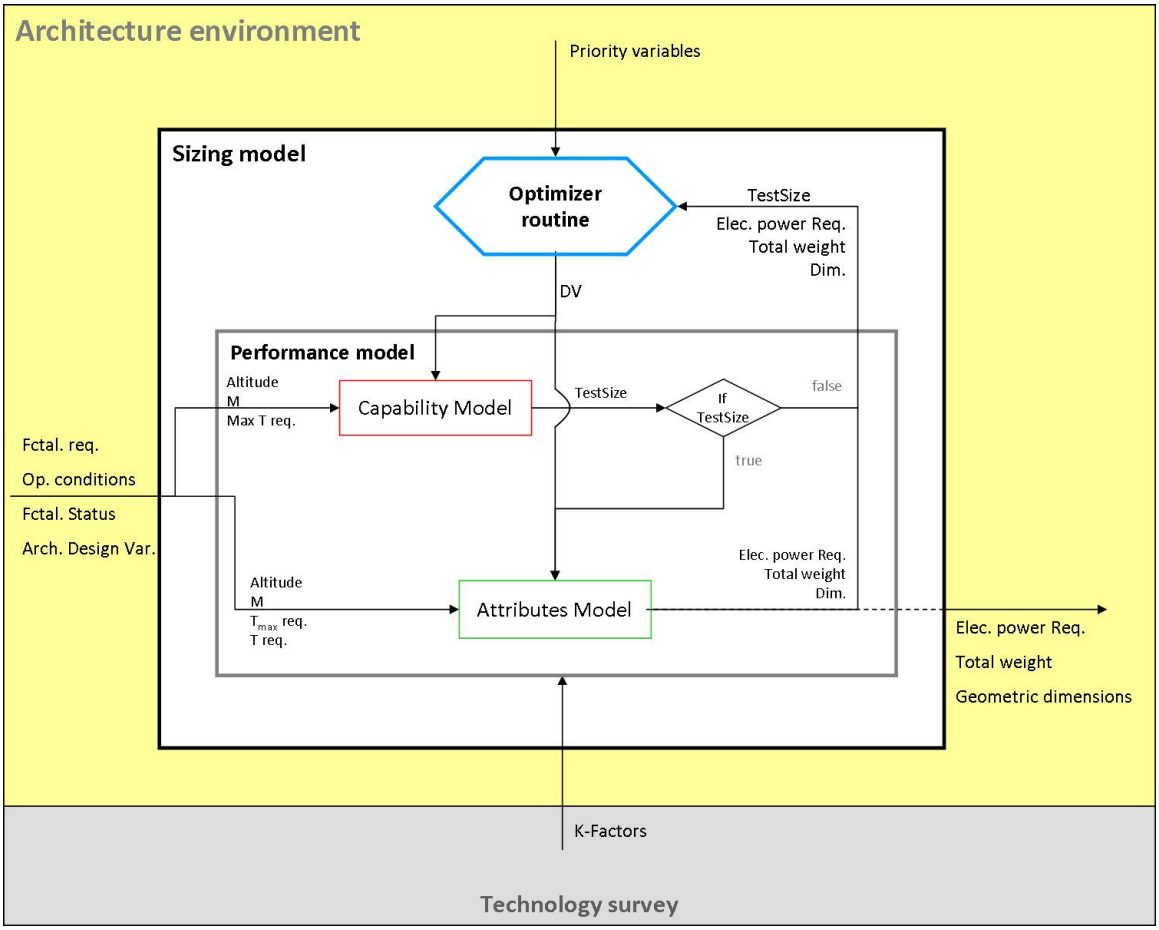


Figure 253: Logic of the electric ducted fan sizing model

The performance model is decomposed into two elements. The first is the capability model which identifies if the design variables (DV) provided by the optimizer routine define an engine capable of providing the thrust required by the mission. The variable "TestSize" is the constraint witness. It caries a signal which specifies whether the engine described by the DV set is capable to provide $T_{max\ req}$ (maximum thrust required). If the signal is valid, the performance model then proceeds to the attribute model which evaluates the attributes of the engine (how much electrical power PowE is required to operate it, how heavy and large the subsystem is). If TestSize is invalid (insufficient thrust capability to satisfy requirements) then the performance model will return TestSize to the optimizer routine and will bypass the attributes model.

In order to better understand how the model is structured, the following structure matrix was used to describe the relationships between the elements constituting the capability and attributes model.

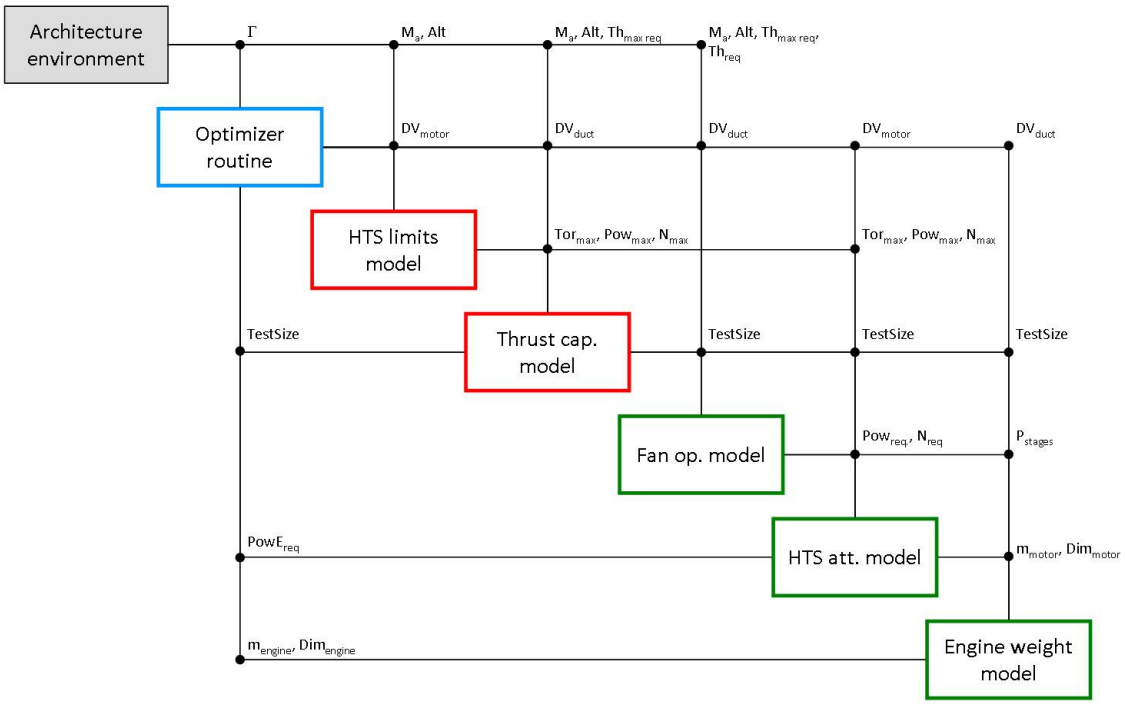


Figure 254: Structure matrix of the electric ducted fan sizing model

In this representation we can see that the capability models (in red) and attribute models (in green) are build around several modules. The capability model is composed the "HTS limits model" and "Thrust cap. model". The first evaluates the maximum shaft power, torque and speed which can be produced by the motor described by the DV. The later uses the description of the fan and its duct to evaluate the thrust capability of the motor/fan/duct assembly (taking into account the power/torque/speed limits of the motor). For each scenario, it will compare the thrust capability to the maximum thrust requirement and will define the value to be assigned to TestSize.

The attribute models are composed of three elements, the first is the fan operational model. It will identify the shaft power and speed necessary to operate the fan along with the pressures in the ducts for all operating points. The "HTS att. model" will

use the shaft powers and speeds to determine the electrical power requirements. Finally the "Engine weight model" will use the pressures and masses and dimensions of the motor/fan/ and duct to determine the weight and major geometrical attributes of the assembly. This appendix will describe the analysis and logic on which these models were constructed. Section B.3 will describe performance model and section B.4 will explain how the optimization routine was constructed.

B.3 Performance analysis

In order to define a descriptive model, it is necessary understand how to perform what the engine community refers to as the "off-design" analysis. This analysis will predict the performance of an engine based on the geometric and performance description of the elements composing it. In the case of our ducted electric fan, the given and unknown parameters are classified and listed in the following table:

Table 89: Overview of input/outputs

Known	Unknown
- Geometrical description of the duct: L_i, A_i, A_2, A_3, $A_{e\text{-}min}$, $A_{e\text{-}max}$, L_e, d_{motor}, l_{motor} - Duct performance characteristics: ΔP_{i2}, ΔP_{3e} - Fan performance characteristics: Nc_{min}, Nc_{max}, $\eta_f = f(Nc,Wc)$, $\pi_{23} = f(Nc,Wc)$ - Motor performance characteristics: Pow_{max} , Tor_{max}, N_{max} - Ambient conditions: M_a, Altitude, ΔISA, $[T_a,P_a,\rho_a] = f$(Altitude, ΔISA)	- Operating settings : N, Wa, Pow - Internal flow properties : M_i, T_i, P_i, ρ_i , M_2, T_2, P_2, ρ_2 , M_3, T_3, P_3, ρ_3 , M_e, T_e, P_e, ρ_e - Subsystem level attributes : $Th_{delivered}$, Pow_{req} m_{engine}

In this section, some variables are referred to as design variables. The design variables will refer to all characteristics of the fan, motor and nacelle which are under the direct control of the designer. The operation-related variables refer to the terms related to throttle settings from the pilot or scheduling settings on the motor controller.

The performance analysis presented herein will be broken down into two parts. The first part is the matching of the fan/motor/duct. This part of the analysis will define the operating settings N, Wa, and Pow. The second part is the aerothermodynamic analysis of the fan/duct which will solve for the internal flow conditions from which engine thrust characteristics can be determined.

This chapter will begin by describing the duct aerothermodynamic analysis which provides a step by step thermodynamic description of the physics of the duct and fan. In section B, it will describe how the fan aerodynamic performance is described using scalable maps. Section C will do the same for the motor, by describing how the design variables were used to determine its limiting operating conditions and performance. Finally, section D will describe the matching process will explain how the duct aerothermodynamic conditions, the aerodynamic performance of the fan and the physical/magnetic limitations of the electric motor are merged and matched to deduce physically sound operating conditions.

B.3.1 Representation of duct aerothermodynamics

This section will describe the resolution process used to describe the overall aerothermodynamic behaviour and performance of the engine. The analysis presented herein will assume that the mass flow and fan operating conditions are known (pressure ratio, speed and efficiency). The definition of these parameters will be described in the following section on matching.

Overview of the duct aerothermodynamics model

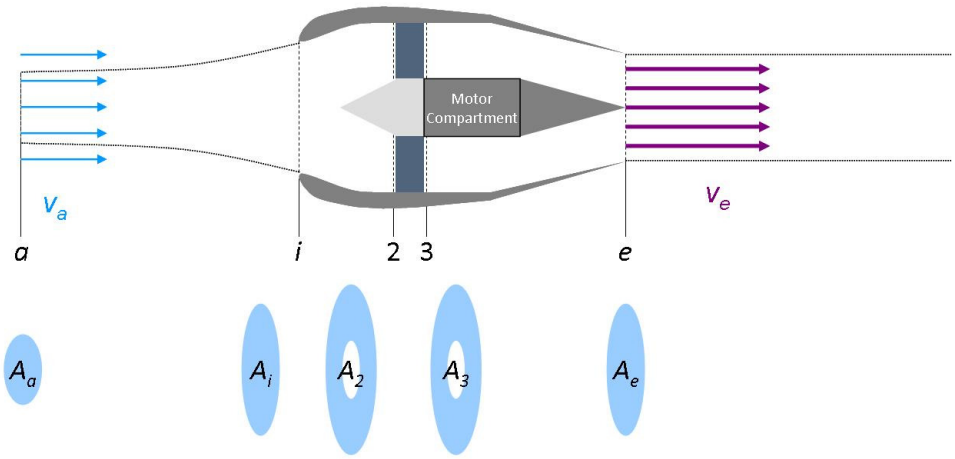


Figure 255: Stations used in the analysis

To facilitate its analysis, the duct is broken down in four sections:

- External streamline compression/expansion (station a – i)
- Inlet compression (station i – 2)
- Fan compression (station 2 – 3)
- Nozzle expansion (station 3 – e)

The engine is operating under conditions specified by the mission. These conditions imply that the inlet is facing some freestream velocity which can be derived from the vehicle true air speed. In order to accommodate the mass flow of air corresponding to a the fan regime, different capture areas (A_a) will occur. As will be described in the following paragraph, the capture area is defined by the mass flow rate required by the engine to operate. The area difference between A_a and inlet throat area A_i will either cause an expansion or compression of the flow.

Once the flow has reached area A_i, it is decelerated based on the geometric constraints imposed by the inlet geometry. Between station 2 and 3 the flow is compressed by the fan. In this section the passage area is assumed to be constant. Once the flow is energized by the fan, the air is expanded in the variable exhaust area nozzle.

The exhaust area of the nozzle (Ae) is adjusted on the fan regime and ambient conditions so that jet is either choked (the exhaust flow is sonic) or fully expanded (the pressure in the jet matches ambient pressure).

The duct aerothermodynamic analysis proposed in herein will assume a mass flow rate. This mass flow rate is itself defined by the fan regime as will be described in the fan-duct-motor matching section.

External flow and inlet flow conditions

The total properties of the air at ambient conditions must be derived from the Mach number specified by the mission and the static properties. The equations necessary to their derivation are available in the last section of this appendix (see equations (119) - (121)). If we assume isentropic 1D flow of a perfect gas, it is possible to solve for the Mach number anywhere in the inlet using the flow parameter (introduced in the section at this end of this appendix).

(70)

$$WFF_n = \frac{W \times \sqrt{To_a}}{A_n \times Po_a}$$

Since the flow parameter can be expressed as a function of the Mach number, it is possible to solve for the Mach number at the inlet throat area using either the proposed inversed function associated with the flow or through an iterative process solving equation (121) for WFF_i.

(71)

$$M_n = arcWFF\left(WFF_i\right)$$

Note: the inversed flow function is presented in equation (122) in the last section of this appendix.

Using this analysis scheme, it is possible to determine the Mach number of the flow at the fan face by substituting A_n by the fan face area A_2 in equation (70) and

solving for M_2 in equation (71). Based on M_2, it is now possible to solve for the static properties at the fan face using isentropic equations (119) - (121).

Note: non-isentropic relationships are available in reference [90]. They could have been used for more accurate analysis (pressure drop from Mach effects, frictions and flow-path angles). For simplicity sake these experimental relationships were not implemented.

Fan

The flow conditions in station 3 are defined by the performance and operation of the fan. The performance of the fan is strongly related to operational-related variables like rotational speed and torque applied on the fan shaft. In order to characterise these relationships, a fan map was used. But at this point we shall assume that the pressure ratio (π_{23} = Po_3/Po_2) and the adiabatic efficiency (η_f) are function of a fan throttle setting referred to as Nc and area of the fan face A_2. The next section will explain how these relationships were derived from the map. But for now, let us consider that π_{23} and η_f can be defined based on equations of the form:

(72)

$$\begin{cases} \pi_{23} = f(Nc, Wc) \\ \eta_{23} = f(Nc, Wc) \end{cases}$$

Using the fan adiabatic efficiency η_{23} and pressure ratio π_{23}, the total pressure and temperature aft the fan are defined as:

(73)

$$Po_3 = Po_2 \times \pi_{23}$$

(74)

$$To_3 = To_2 \times \left[1 + \frac{1}{\eta_{23}} \left(\pi_{23}^{\frac{\gamma-1}{\gamma}} - 1 \right) \right]$$

Exhaust system

For simplicity sake, 1 dimensional-isentropic-perfect gas assumptions were used for the analysis of the nozzle. More sophisticated (also more accurate) experimental solutions can be found in reference [91]. Based on the isentropic assumptions, the total properties of the exhaust flow are identical to those derived in station 3.

The design nozzle design that will be considered by the model has a variable exhaust area. This variable exhaust area allows for increasing the efficiency of the fan by tuning independently the mass flow and speed of operation of the fan (see section on matching for more details). This variability also allows for a better control of the chocked condition (sonic exhaust).

Several boundary conditions may occur. The first will be the subsonic exhaust conditions and the second the sonic condition. Since the nozzle is a strictly convergent nozzle, supersonic exhaust conditions are not applicable. For a given scenario, the applicable boundary condition can be identified from the "fully-expanded Mach number" (M_{ex}). This Mach number corresponds to the exhaust mach number to which the flow would need to be accelerated in order to match the jet and ambient static pressures. Given the ambient static pressure, and the total pressure in the nozzle, M_{ex} can be determined by transforming the isentropic equation (120):

(75)

$$M_{ex} = \sqrt{\frac{2}{\gamma - 1}\left[\left(\frac{Po_e}{P_a}\right)^{\frac{\gamma}{\gamma-1}} - 1\right]}$$

If M_{ex} is less than one, the nozzle will adopt an exhaust area which will allow the flow to be accelerated to M_{ex}. Otherwise, the nozzle will accelerate the flow to sonic conditions (the highest Mach condition achievable for a convergence nozzle). The resolution of the nozzle aerodynamics is described for each of these boundary conditions. The subsonic exhaust requires two boundary conditions:

- The static pressure of the jet has to match ambient pressure ($M_e = M_{ex}$)

- The laws of mass and energy continuity and conservation of mass have to be respected

Since the static pressure of the jet matches the ambient pressure, we know that the exhaust Mach number must match M_{ex} from Equation (75). Using this Mach number, we can derive the static temperature:

(76)

$$T_e = To_3\left(1+\frac{\gamma-1}{2}M_e\right)^{-1}$$

Using the static temperature, it is now possible to determine the jet speed:

(77)

$$v_e = M_e \times \sqrt{\gamma \times R \times T_e}$$

The nozzle area which will allow generating these conditions can be derived by adapting the relationship implied by the flow parameter:

(78)

$$A_e = \frac{Wa}{Po_3 \times M_e}\sqrt{\frac{To_e \times R}{\gamma}}\left(1+\frac{\gamma-1}{2}M_e^2\right)^{\frac{\gamma+1}{2(\gamma-1)}}$$

The sonic exhaust requires two boundary conditions:

- The Mach number at the exhaust is equal to unity
- The laws of mass and energy continuity and conservation of mass have to be respected

Since the Mach number is equal to 1, the flow properties can be derived from the total properties.

(79)

$$T_e = To_3\left(1+\frac{\gamma-1}{2}\right)^{-1}$$

(80)

$$P_e = Po_3\left(1+\frac{\gamma-1}{2}\right)^{-\frac{\gamma-1}{\gamma}}$$

Based on the static temperature, the jet speed can be determined as:

(81)

$$v_e = \sqrt{\gamma \times R \times T_e}$$

The nozzle area which will allow generating these conditions can be derived by adapting the relationship implied by the flow function:

(82)

$$A_e = \frac{Wa}{Po_3}\sqrt{\frac{To_e \times R}{\gamma}}\left(1+\frac{\gamma-1}{2}\right)^{\frac{\gamma+1}{2(\gamma-1)}}$$

Thrust and summary of propulsive system

The net thrust produced by the engine is defined by the following equation:

(83)

$$Fn = \dot{m}\left(v_e - v_a\right) + A_e\left(P_e - P_a\right)$$

The derivation of this equation can be found in [92]. This expression of thrust is composed of two elements. The first is the momentum contribution to thrust and accounts for the difference in momentum between the flow upstream and the jet out of the exhaust. The second element accounts for the thrust resulting from the pressure difference between ambient and exhaust pressure. This pressure difference will occur in situations where the exhaust flow is choked. In this situation the pressure at the nozzle is larger than the ambient pressure. This pressure differential over the exhaust surface contributes to thrust.

Limiting conditions

The largest mass flow through a given area corresponds to the flow at sonic condition. This conditions can be expressed using by the flow parameters corresponding to M=1.

(84)

$$WFF(1) = \frac{W \times \sqrt{To}}{A \times Po} = \sqrt{\frac{\gamma}{R}} \times \left(1 + \frac{\gamma - 1}{2}\right)^{-\frac{\gamma+1}{2(\gamma-1)}} = 0.040425 \left[\frac{\text{kg.}\sqrt{\text{K}}}{\text{N} \cdot \text{s}}\right]$$

Therefore the limiting mass flow that can realistically pass through the duct corresponds to the mass flow which will first generate the choked condition described by equation (84). Assuming that Po and To are constant in the inlet and nozzle, we can deduce that choked conditions may occur at the smallest area of either zones. For the inlet this correspond to station i (the inlet throat) and for the nozzle to station e (the exhaust).

Implementation

Equations (70) through (84) were implemented in the Matlab function `FanModel` presented page 572.

B.3.2 Fan performance

A fan is device transforming mechanical energy from a shaft into flow potential energy. The amplitude of the potential energy increase is described by the ratio of flow total pressures at stations around the fan (π_{23}). In order to characterize the quality of this transformation process an addiabatic efficiency factor (η_{23}) is used to qualify the ratio between the hypothetical work necessary to perform the compression in an ideal (isentropic) fashion and the work actually necessary to operate the fan (assuming that no heat is lost during the compression process – hence adiabatic).

The fan transformation in energy is achieved by the aerodynamic loads generated by a set of blades rotating in the flow of air. Therefore Hill and Peterson [92] consider that the factors influencing π_{23} and η_{23} are:

(85)

$$[\pi_{23}, \eta_{23}] = f(Wa, Po_2, To_2, N, \gamma, R, \upsilon, design, D)$$

Note: The term ν refers to the kinetic viscosity factor of the flow, "design" makes a reference to the geometric shape of the fan while the term D refers to the diameter (i.e its scale)

By performing a Buckingham-Pi analysis [92], it is possible to reduce from 9 to 5 the number of independent terms.

(86)

$$[\pi_{23}, \eta_{23}] = f\left(\frac{Wa\sqrt{\gamma \times R \times To_2}}{Po_2 \times D^2}, \frac{N \times D}{\sqrt{\gamma \times R \times To_2}}, \frac{N \times D^2}{\upsilon}, \gamma, design\right)$$

Therefore if we want to characterize the performance of a scalable fan (with a fixed "design"), over a range of conditions where the Reynolds number effects are negligible and where the gas constant R and ratio of specific heats γ are constant, it is now possible to further reduce the number of independent terms:

(87)

$$[\pi_{23}, \eta_{23}] = f\left(\frac{Wa\sqrt{To_2}}{Po_2 \times D^2}, \frac{N \times D}{\sqrt{To_2}}\right) \text{ or } f\left(\frac{Wa\sqrt{To_2}}{Po_2 \times A_2}, \frac{N \times D}{\sqrt{To_2}}\right)$$

In order to adjust for ambient conditions, the terms θ and δ are commonly used by the community. These terms are defined as follows:

$$\begin{cases} \theta = {To_a}/{To_{ref}} \\ \delta = {Po_a}/{Po_{ref}} \end{cases} \quad \text{with} \quad \begin{matrix} To_{ref} = 288.15 \text{ K} \\ Po_{ref} = 101.325 \text{ kPa} \end{matrix}$$

Assuming that the total temperature and pressures are conserved through the inlet and assuming that we are considering a fixed scale (D) we can therefore assume that π_{23} and η_{23} are depended on two factors:

(88)

$$[\pi_{23}, \eta_{23}] = f(Wc, Nc) \quad \text{with} \quad \begin{cases} Wc = \dfrac{Wa\sqrt{\theta}}{\delta} \\ Nc = \dfrac{N}{\sqrt{\theta}} \end{cases}$$

The fan map is an array of data that qualifies the performance of the fan stage based on this assumption that performance is characterized by π_{23} and η_{23} as a function for Wc and Nc.

As the fan is scaled, it is essential that the proper correction is made on Wa and N, in order to refer to the point in the map which represents the relevant aerodynamic phenomenon. Therefore similarly to correction factors θ and δ which adjusted for variations in ambient condition, it is necessary to define scaling factors capturing changes in aerodynamic phenomenon due to scale.

In order to conceptually understand the role of these scaling factors, we shall use the analogy between the aerodynamic phenomenon around a wing profile and a fan blade. It is generally assumed that aerodynamic loads on a given wing profile can be characterized as a function of free-steam velocity and angle of attack (Buckingham–Pi analysis similar to the on presented in the previous section). In the same fashion the aerodynamic loads over fan blade which can be seen as a continuous series of wing profile can be characterised by scaled mass flow and rotational speed.

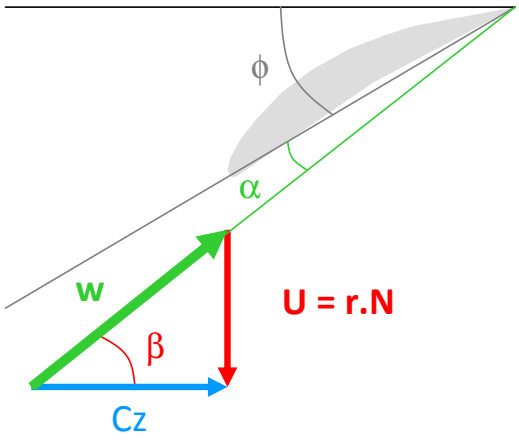


Figure 256: Incoming flow velocity triangle

From Figure 256, we can see that the aerodynamic loads over the blade are driven by the amplitude of w and α. We can also say that w and α can be derived from the rotational speed U and Cz:

(89)

$$w(r) = \sqrt{Cz^2 + (r.N)^2} \quad \text{and} \quad \alpha(r) = \tan^{-1}\left(\frac{Cz}{(r.N)}\right) - \phi(r)$$

Since the shape of the blade is maintained through the sizing process, the performance characteristics like coefficients of lift and drag are constant (Reynolds number effects assumed constant). Therefore the loads upon (or caused by) the blades are determined by w and α. Therefore the pressure ratio and efficiencies (which are independent from the scale) are directly related to w and α. Since a map is not parameterized with respect to w and α, it is necessary to understand how their relationships to Wa and N change with respect to scale. In order to introduce the term Wa in equation (89), let us consider the mass flow passing through the fan area A_2.

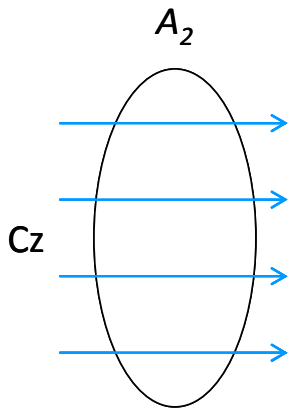


Figure 257 : Mass flow through fan area

Since Cz is the normal component of the flow speed with respect to the fan area, Cz and mass flow can be related by the following equation.

(90)

$$Wa = \rho \times Cz \times A_2$$

By substituting this expression back into equation (89), we now have the following expressions for w and α:

$w(r) = \sqrt{\left(\frac{Wa}{\rho A}\right)^2 + U(r)^2}$ and $\alpha(r) = \tan^{-1}\left(\frac{Wa}{\rho \times A \times U(r)}\right) - \phi(r)$ Since through the sizing process, the fan is photographically scaled, the flow characteristics (w and α) are expected to be maintained for all radial position ($\bar{r} = r / r_{tip}$). By substituting $\bar{r}$ into expression (89), we get:

(91)

$$w(\bar{r}) = \sqrt{\left(\frac{Wa}{\rho A}\right)^2 + (\bar{r} \times N \times r_{tip})^2} \quad \text{and} \quad \alpha(\bar{r}) = \tan^{-1}\left(\frac{Wa}{\rho \times A \times N \times r_{tip}} \times \frac{1}{\bar{r}}\right) - \phi(\bar{r})$$

In order for both w and α to be consistent for all values of $\bar{r}$ and given any value of A_2 and r_{tip}, it is necessary to conserve the value of Cz and $U.r_{tip}$. This observation induces the need for the following scaling factors A/A_{ref} and $r_{tip}/r_{tip\text{-}ref}$:

(92)

$$Wcs = \frac{Wa\sqrt{\theta}}{\delta}\frac{A_{ref}}{A} \quad \text{or} \quad Wa = \frac{Wcs \times \delta}{\sqrt{\theta}}\frac{A}{A_{ref}}$$

and

$$Ncs = \frac{N}{\sqrt{\theta}}\frac{r_{tip}}{r_{tip-ref}} \quad \text{or} \quad N = Ncs\sqrt{\theta}\frac{r_{ref}}{r_{tip}}$$

These scaling factors along with the operation related correction factors δ and θ make the corrected-scaled speed and mass flow (designated by the suffix "cs"). The map used in this work was defined experimentally in reference [93]. The geometry of the reference fan corresponds to the E^3 engine' fan and its dimensions are described in the following figure:

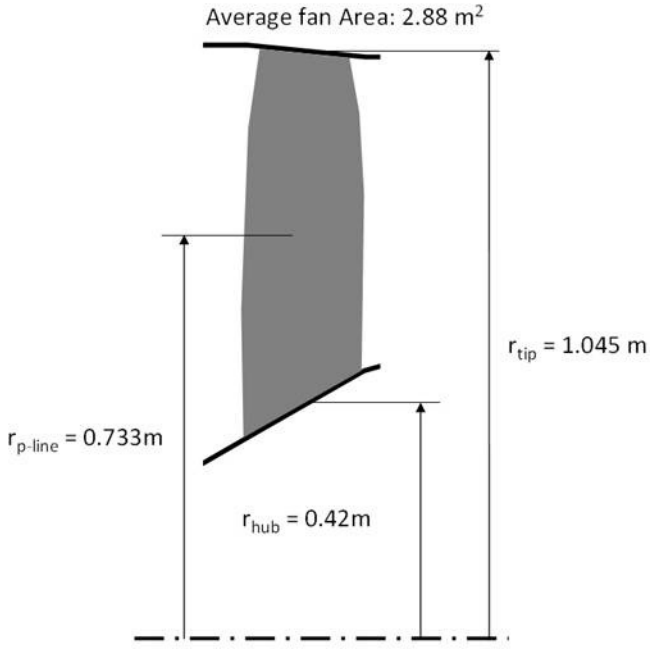


Figure 258: Baseline fan geometry

Constraints imposed by the fan

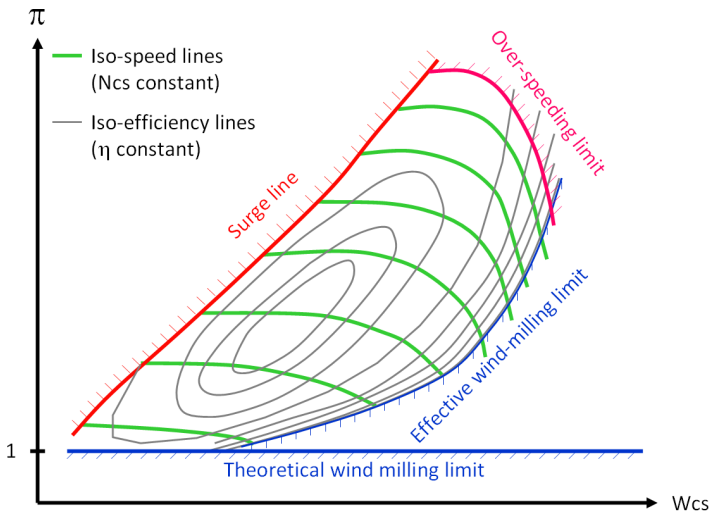


Figure 259: Notional fan map

The fan operation is limited to a region bounded by positive stall (or surge) and what will be refered to as wind-milling. If we follow the green lines in Figure 259, we can see that for a given rotational speed, as the amount of air fed to the fan decreases (or as Wcs decreases), the operation is suddenly interrupted by the surge line. The physical meaning of the surge line can be understood by local characteristics of the flow over the blade. As Wcs decreases Cz also decreases which, for a given rotational speed augments the amplitude of the angle of attack. Once the angle of attack reaches the stall angle, the compressor is unable to compress the air (i.e is unable to operate).

On the other hand “wind-milling” refers to the opposite phenomenon. Textually wind-milling refers to conditions where the fan is extracting energy from the flow (rather than injecting). In wind-milling mode, the flow rate across the fan face (or Cz) is too large with respect to the rotational speed N. In this condition, the angle of attack on a part of the blade (starting at the hub) will be facing an apparent flow at a negative angle of attack. This condition will drastically decrease the efficiency of the fan. Generally these situations are not characterized by the fan map.

The third constraint limits the speed at which the fan is expected to operate. This over-speeding limit can be the expression of structural limitations of the fan or can result from the formation of sonic shocks on the blade which may compromise its efficiency (or worse damage the blade).

Figure 259 highlights the constraints described earlier. In the context operational constraints, we can see that the definition of wind-milling will be slightly extended to qualify the ensemble of operating conditions where the fan efficiency quickly drops due to the apparitions of local negative angles of attacks caused by the excessive flow speed.

B.3.3 Motor performance

The motor performance analysis was derived from an existing model defined by Philippe Masson et al. in support for publication [reference to Philippe and Taewoo's paper]. The physics underlying this model were described in this reference and will not be discussed further. The model originally coded in excel was transcribed into Matlab. This transcription is presented below. Most of the terminology used in the analysis is explained as comments.

```
function [PowReq,W_totalPhilippe,Dimensions] = HSTmotor2(PowShaft,NShaft,DVMotor)

% Model based on Philippe Masson model
% Translated and addapted by Cyril de Tenorio

Testingmode=0;      % should be 0 unless you are testing the function
%% for testing Make sure you comment the function line
if Testingmode==0
    %% Input from the fan
    clc
    clear all
    PowShaft = 2705 * 10 ^3;    % Power extracted on the shaft by fan [W]
    NShaft = 13000;              % Speed of the shaft [rpm]

    %% Design variables
    Ks = 299.6;                 % Armature Ampere turn loading (kA/m)
    ARm = 2.13;                 % Machine shape factor (La/ro)
    eis = 45.4;                 % Electrical radial airgap (mm) (5 mm is pretty small)
    Ispe = 93.725879;           % Armature current density (Arms/mm2)
    n2p = 4;                    % Number of pole pairs
    ToP = 20.0;                 % Temperature (K)
    ro = 153;                   % Mean armature radius  ro (mm)
else
    Ks = DVMotor(1);            % Armature Ampere turn loading (kA/m)
    ARm = DVMotor(2);           % Machine shape factor (La/ro)
    eis = DVMotor(3);           % Electrical radial airgap (mm) (5 mm is pretty small)
    Ispe = DVMotor(4);          % Armature current density (Arms/mm2)
    n2p = DVMotor(5);           % Number of pole pairs
    ToP = DVMotor(6);           % Temperature (K)
    ro = DVMotor(7);            % Mean armature radius  ro (mm)
```

```
end
% Set by default
eisMin = 15;                % Minimum electrical airgap (mm)
te = 8;                     % Armature backiron radial standoff from armature (mm)
Jco = 150;                  % Critical current density (77K, 0T) Jco (A/mm2)
HSTopF = 0.6;               % HTS operating factor
HSTfr = 0.8;                % HTS filling ratio
beta = 120*pi/180;          % HTS angular aperture (rad)
PhiBI = 2.2;                % Magnetic flux density in the back iron (T)
CryEff = 30;                % Carnot efficiency (cryocooler) [%]
SpeWCryGVB=5/1643.9868;     % GVB cryocooler [kg/W]
HSTden = 8500;              % HTS density (kg/m3)
Armden = 8500;              % Armature density (kg/m3)
PhiL = 1.28;                    % Iron specific losses GVB (1.5 T, 50 Hz) 4 mil Hiperco
50A(W/kg)

ArmOpFactor = 0.6;          % Armature operating factor
ArmLength4loss  = 12;       % Armature AC loss dimension (µm,mils)

Steelden = 8321.369721;                 % density of steel     (kg/m3)
Steelstrain = 0.1;                      % strain                (%)
SteelModulus = 30000000*6894.75729;     % modulus               (psi)
Sigma_Op = Steelstrain/100*SteelModulus;% operating stress      (N/m2)

%% Calculations
TorShaft = 30000*PowShaft/1000/pi/NShaft;   % Torque (Nm)
es = pi/3*Ks/Ispe;                        % Armature thickness es (mm, in)
freqE =NShaft*n2p/60;                       % Electrical frequency (Hz)

%% Geometry
% HTS (rotor) outside radius (mm) r2
r2 = ro-eis-es/2;

% Back iron inner radius (mm) rs
rs = ro + es/2 + te;

% Active length (mm) La
La =ARm*ro;

% Total length (mm) Ltot
Ltot = La+pi*ro/n2p;

%% Electro Magnetic analysis
% Tangentail Speed at r2 (m/s)
v_r2 = r2/1000*2*pi*NShaft/60;

% Iron specific losses (BFe, f) (W/kg)
BFe= PhiL/2*(freqE/50 + (freqE/50)^2)*(PhiBI/1.5)^(2);


% Radial no load field at armature Bro (T)
Bro = 1000000*TorShaft/(1.4142*Ks *pi*((ro^3)*ARm));

% Radial max field on HTS winding (T)
BrWind = Bro*(ro/r2)^(n2p+1)*(1+(r2/rs)^(2*n2p))/(1+(ro/rs)^(2*n2p));

% Critical current density (based on max radial field) (A/mm2)
JcMax = Jco*(5.22-1.5166*BrWind+0.353*BrWind^2-...
    0.037625*BrWind^3+0.0014773*BrWind^4)*(1.4235-0.022467*ToP+...
    0.000058307*ToP^2);
% Field winding current density (A/mm2)
Jf = HSTopF*HSTfr*JcMax;

% Intermediate calculation for r1
r1temp = Bro*(n2p+2)*(ro/r2)^(n2p+1)/(0.0008*Jf*r2*sin(beta/2))/...
    (1+(ro/rs)^(2*n2p));

% Rotor HTS inside radius (mm) r1
r1 = r2*(1-r1temp)^(1/(n2p+2));
% Rotor HTS mean radius (mm)
% r12 = (r2+r1)/2;
% Synchronous reactance (p.u.)
SynReac = 0.0004*pi*Ks*(1+(ro/rs)^(2*n2p))/(1.4142*Bro);

% Back iron thickness (mm)
tiron =2*rs*Bro*(ro/rs)^(n2p+1)/n2p/(1+(ro/rs)^(2*n2p))/PhiBI*...
    (1+SynReac*SynReac)^.5;

% Back iron external radius re  (mm)
re = rs+tiron;
```

```
% Back iron weight (kg)
W_iron= 0.00000765*pi*(re*re-rs*rs)*La;


%% Losses
% Iron losses (W)
Loss_iron = W_iron*BFe;

% Armature Required? critical current density (A/mm2)
Jca =(Ispe/ArmOpFactor)*1.414;

% Volume of superconductor (m3) !Intermediary calculation added by Cyril
Vsc = 0.000000002*pi*ro*es*(La+5*pi*pi/12/n2p*ro);
% Arm. AC losses (W) (at cold T)
Loss_Arm= 8/3/pi*Jca*ArmLength4loss*freqE*Bro*Vsc;

% Cooler Input Power for Armature AC losses (W)
Loss_cool= Loss_Arm *(300-ToP)/ToP/(CryEff /100);

% Thermal losses (W)
Loss_th = 2;

% AC field winding losses (W)
Loss_field = 0;

% Total field winding cryogenic losses (W)
Loss_wind = Loss_th+Loss_field;

% Total electrical losses (W)
Loss_TotElectic = Loss_cool+Loss_iron;

% Efficency = 1-Loss_TotElectic/PowShaft;

PowReq = PowShaft + Loss_TotElectic;

%% Torque tube
% incr. ms/MrotorHTS    ( - )
IncrMshaft =(Steelden*(r2/1000)^2*((NShaft/60)*2*pi)^2)/Sigma_Op;

% Total str ms/MrotorHTS( - )
Sigma_tot = IncrMshaft/(1-IncrMshaft);

% Torque tube mass      (kg)
SingleTorTubeW = (((Ltot+2*r2)/1000)*TorShaft*Steelden)/(Sigma_Op*r2/1000);

%% Weight Calculations
% Weight of torque tubes (2) (kg)
W_TorTube = 2 * SingleTorTubeW;

% Armature winding weight (kg)
W_wind = Armden*Vsc;

% Rotor HTS weight (kg)
W_rotor= HSTden*HSTfr*beta*(r2*r2-r1*r1)*(La+pi/2*(pi-beta/2)*...
    (r2+r1)/2/n2p)*0.000000001;

% Weight HTS total (kg)
W_HST = W_wind + W_rotor;

% Weight of rotor containment (kg)
W_rotorCont = W_rotor *Sigma_tot;

% Weight Motor/Gen Total (kg)
W_Motor = W_HST + W_iron + W_rotorCont + W_TorTube;

% Cryocooler mass (kg)
W_cryo = 157*(2.7183)^(-0.0533*ToP)*(Loss_Arm+Loss_wind)^(0.009*ToP+0.1275);

% GVB cryocooler weight (kg)
W_cryoGVB = Loss_cool*SpeWCryGVB;

% Weight Motor + Cryoc (kg)
W_MotorCry = W_Motor + W_cryoGVB;

% Torque tubes (2) combined thickness (m)
t_tortube = 2 * (TorShaft/(2*pi*(r2/1000)^2*Sigma_Op));

% Containment tube thickness (mm)
t_cont = (W_rotorCont/Steelden)/(pi*2*r2/1000*Ltot/1000)*1000;
```

```
% Weight of Motor including 60% structure weight (kg)
W_MotorStruct = (W_HST + W_iron)*1.6 + W_rotorCont + W_TorTube;

% Total weight including cooler (kg)
W_total = W_MotorStruct+W_cryoGVB;

% Weight as calculated by Philippe model
W_totalPhilippe = W_MotorCry +(W_HST+W_iron)*1.6 - (W_Motor);

Dimensions(1,1) = Ltot;
Dimensions(2,1) = re;

if ARm<2*pi/n2p
    Error =1;
elseif  v_r2<0
    Error =2;
elseif Bro<0 || Bro>1.3
    Error = 3;
elseif BrWind <0 || BrWind>3.5
    Error = 4;
elseif r1temp>0.85
    Error = 5;
elseif re>600
    Error = 6;
elseif eis<eisMin+t_cont+r2/10
    Error=7;
else
    Error = 0;
end
if Error>0
    PowReq = 0;
    W_totalPhilippe = 0;
    Dimensions = zeros(2,1);
end
```

This model was originally defined as an "on-design" sizing analysis. In other words, given an operating condition and some key design variables, this model would size the motor and estimate its performance. Given the fact that the electric fan model will have to be sized for an ensemble of operating conditions, this model has been adapted. The analysis underlying the code above was maintained but the code was adapted to enable multi-design operational (e.i. "off-design") analysis.

It is important to note that the model above is based on seven design variables:

- The ampere turn loading (Ks)
- The machine shape factor (ARm)
- The electrical radial airgap (eis)
- The armature current density (Ispe)
- The number of pole pairs (n2p)
- The internal temperature of operation (ToP)
- The mean armature radius (ro)

Based on these seven design variables and 2 operation condition specifications (shaft power - Pow and shaft speed - N), the attributes of the motor can be defined by the model. It is important to note that not any operating conditions are feasible. Physical limits are captured by seven constraints presented in the last lines. The first constraint is a geometrical constraint limiting the aspect ratio of the machine as a function of the number of pole pairs:

(93)

$$ARm > \frac{2\pi}{n2p}$$

The second constraint is also geometrical but translates into a relationship between ro, Ks and Ispe:

(94)

$$v_{r2} > 0 \qquad \Rightarrow \qquad ro > eis - \frac{\pi}{6}\frac{Ks}{Ispe}$$

The third and fourth constraints results from magnetic limitations and are constraining the maximum amount of torque that can be produced by a given motor:

(95)

$$B_{\mathrm{ro}} < B_{\mathrm{ro}}\big|_{\max} = 1.3 \qquad \Rightarrow \qquad Tor_{\max} < \frac{B_{\mathrm{ro}}\big|_{\max}}{C_{ro}}$$

and

(96)

$$B_{\mathrm{rwind}} < B_{\mathrm{rwind}}\big|_{\max} = 1.3 \qquad \Rightarrow \qquad Tor_{\max} < \frac{B_{\mathrm{rwind}}\big|_{\max}}{C_{ro} \times C_{om}}$$

With $Cro = \frac{\left[10^6\right]}{\left(\sqrt{2} \times Ks \times \pi \times ro^3 \times ARm\right)}$ and $C_{om} = \left[\left(\frac{ro}{r_2}\right)^{n2p+1}\left(1 + \frac{\left(r_2/r_s\right)^{2 \times n2p}}{1 + \left(ro/r_s\right)^{2 \times n2p}}\right)\right]$

The fifth constraint limits the value that the ratios (r1t) of the thickness to radius of the HTS field winding:

(97)

$$r1t = \left(\frac{r_2 - r_1}{r_2} \right)^{n2p+1} < r1t\big|_{\lim} = 0.85$$

By developing the expression of r1t as function Tor_{max}, the constraint is a non linear constraint on Torque:

(98)

$$\frac{Tor_{\max}}{\sum_{i=1}^{4} a_i \times Tor_{\max}^{i}} < \frac{r1t\big|_{\lim} \times C_{jf}}{C_{ro} \times C_{1t}}$$

With $C_{jf} = \frac{Jc_0 \times HTSopF \times HTSfr}{a + b \times ToP + c \times ToP^2}$ and $C_{1t} = \frac{(n2p+2)\left(\frac{ro}{r_2}\right)^{n2p+1}}{r_2 \times \sin\left(\frac{\beta}{2}\right)} \left[1 + \left(\frac{ro}{r_s} \right)^{2 \times n2p} \right]^{-1}$

There are no explicit solutions for this inequality on Tor_{max}. In order to solve for the maximum torque that this system can provide, equation (98) is solved iteratively using the largest Tor_{max} feasible on constraint three, four and six. Constraint six is another geometrical restriction which limits the back iron radius to 600mm. This constraint directly imposes a limitation on the maximum torque:

(99)

$$r_e < r_e\big|_{\lim} = 600 \qquad \Rightarrow \qquad Tor_{\max} < \frac{\sqrt{\left(r_e\big|_{\lim} - r_s\right)^2 - C_{sr}^2 C_{ti}^2}}{C_{ro} C_{ti}}$$

With $C_{sr} = \left[\frac{0.0004\pi}{\sqrt{2}} \right] Ks \left[1 + \left(\frac{ro}{r_s} \right)^{n2p+1} \right]$ and $C_{ti} = \frac{2 \times r_s \left(\frac{ro}{r_s} \right)^{n2p+1}}{\Phi_{Bi} n2p \left[1 + \left(\frac{ro}{r_s} \right)^{n2p+1} \right]}$

The seventh and last constraint results from a geometrical constraint which requires that the containment tube for the rotor winding must fit within the electrical airgap. The containment tube is sized as a function of the speed of rotation. Therefore this constraint determines the speed limit of the motor.

(100)

$$t_{cont} < t_{cont}\big|_{\max} = eis - eis\big|_{\min} - r_2/10 \qquad \Rightarrow \qquad N_{\max} < \sqrt{\frac{t_{cont}\big|_{\max}}{C_{mM}\left(t_{cont}\big|_{\max} + C_{tcont} W_{rotor}\right)}}$$

With $C_{tcont} = \frac{\left[1000^3\right]}{2\pi \times r_2 \times L_{tot} \times \rho_{steel}}$ and $C_{mM} = \frac{r_2^{\,2} \times \rho_{steel}}{2\pi \times r_2 \times L_{tot} \times \rho_{steel}} \left[\frac{2\pi}{60} \frac{1}{1000^2}\right]$

Important Note: None of the constraints could be traced back to the power limit of the motor. In order to maintain the consideration of the power limit in the model a temporary power limit was defined using Tor_{max}:

(101)

$$Pow_{\max} = 0.5 \times Tor_{\max} \times 5114 \times \left[\frac{2\pi}{60}\right]$$

The calculation above is based on the max necessary speed from the fan at M=0.5 alt=40,000ft (5114 RPM). The arbitrary coefficient 0.5 corresponds to the limit on power coming from some unidentified physical phenomenon in the motor.

Motor limit model

Based on the resolution process of the constraints in Masson's model, a motor limit model was defined: `HSTlimits`. This model will identify the largest Pow_{max}, Tor_{max} and N_{max} feasible (i.e. which will fall within the seven constraints described above). The Matlab function is reproduced page 569. This function is the embodiment of the model element "HTS limits model" in Figure 254.

The motor model was only a slight modification from the model by Masson. The single operation point characteristics for power and speed were replaced by the maximum torque (Tor_{max}) and speed (N_{max}). These limiting characteristics were used in conjunction with the design variables to define fixed motor attributes (geometric dimensions and weights). The operational attributes like heat rejections, efficiencies and power requirements were defined for each scenario by using the amount the shaft power and speed required by the fan. The maximum heat rejection across the mission is used to estimate the size of the cryocooler. This model was implemented in the Matlab function `HTSatt`. The code associated with this function is available at the end of this appendix page 580.

B.3.4 Duct-Fan-Motor matching

In the context of performance modelling, the term "matching" refers to the synthesis of performance models of the sub-elements composing the engine so that their operation are physically compatible. In the context of the electric ducted fan these elements are:

- The duct
- The fan
- The electric motor

The matching process implies that several operational variables must be "matched". In the context of this engine these matching operational variables are:

- The flow passing through the fan must match the flow through the duct (inlet & nozzle)
- The mechanical balance between the fan and the engine must be respected which implies that:
- The power required by the fan must match the power delivered by the motor (adjusted to shaft and gearbox inefficiencies)

- The speed at which the fan is rotating must match the rotational speed of the motor (with applicable adjustments in case a gear-box was used)

In addition to these matching constraints we saw that the operations of the sub-elements were constrained. Therefore the matching process is a delicate analysis within a multi-constrained operational space. Based on the relationships described in the previous sections a system of equations was constructed to present an overview of the matching problem:

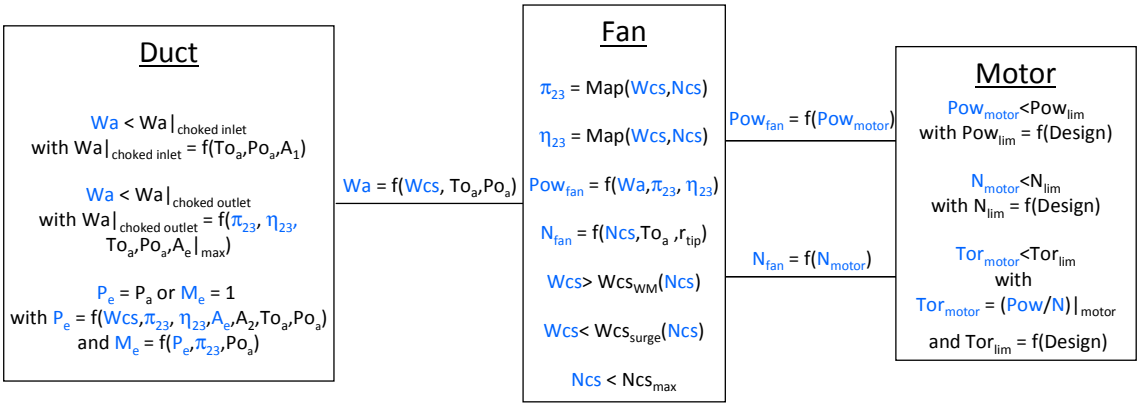


Figure 260: Matching problem

In this figure the variables represented in blue are the undetermined terms that must be solved for. Overall there are thirteen undetermined terms, for eleven equations and eight inequality constraints. Two of the thirteen terms can be independently set and could be considered as throttle settings (N_{motor} and A_e). One can see the control of these parameters as the throttle effects on the fan regime (N_{motor}) and the mass flow (A_e). Therefore, the matching process must solve for the values of the remaining eleven undetermined variables).

For simplicity sake, the matching process implemented in this thesis will not consider the term Ae as independent. Instead it shall assume that the fan regime will follow an operating line. This operating line assumes that there is a relationship between the fan speed and the mass flow. Since the mass flow is set by the exhaust area and the pressure ratio, we can assume that the matching process implemented corresponds to an

engine where the variable exhaust area is set by a scheduler which will force a fixed relationship between the regime and the mass flow through the engine.

This assumption was motivated by the fact that using Ae as an independent factor would require inverting equations (78) to (82) which can not be performed analytically. Therefore solving this matching problem would require a computationally expensive recursive process necessary to both pick a point in the feasible fan operating space (i.e. optimization would be necessary) and the iterative process necessary balance the pressure ratio and mass flow imposed by Ae. These additional processes would be computationally expensive and would severely burden the evaluation of the engine performance as they would need to occur for each operational point considered.

In the following paragraph we will first observe how the various constraints imposed by the fan, motor and duct will shape the operational envelope of the assembly. The next paragraph will present the approach used in the resolutions of the matching process.

Motor/fan constraints

In order to get a fell for the constraints implied by the motor/fan matching we will consider the operational envelope on their mutual maps. As discussed earlier the fan operation is constrained by its surge constraint (pink in the fan map), the wind milling limit (blue) and max speed (purple). The motor is constraint by its torque constraint (horizontal orange line in the motor map), the power limit (parabolic constraint) and max speed (vertical orange line). It is important to note that all fan constraints are corrected to the operating conditions while the motor constraints are not. Therefore the feasible operation space will change drastically in shape depending on the flight speed and ambient conditions.

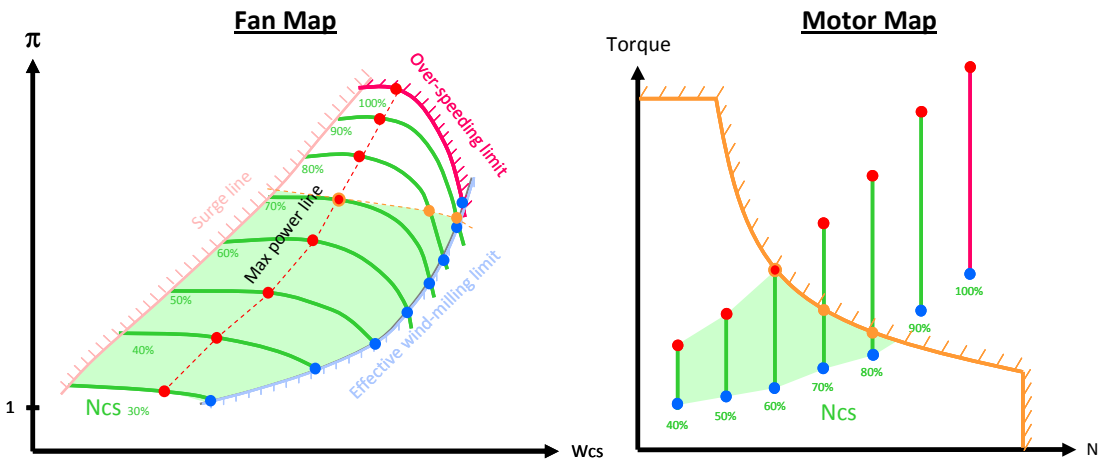


Figure 261: Operational fan/motor matching

On the left hand side, we can see the fan map with iso-speed lines. Each line imply a range of power requirement from the shaft. This range is more explicitly presented on the right hand side on the motor map. For each fan speed (vertical lines in the motor map), the highest torque value is achieved for some mass flow which yields a pressure ratio, mass flow and efficiency combination maximizing the fan power equation:

(102)

$$Pow_{fan} = \frac{Cp}{\eta_{23}} To_2 \left(\pi_{23}^{\frac{\gamma-1}{\gamma}} - 1 \right) \times Wa$$

Similarly the torque equation corresponds to:

(103)

$$Tor_{fan} = \frac{Cp}{\eta_{23}} To_2 \left(\pi_{23}^{\frac{\gamma-1}{\gamma}} - 1 \right) \times \frac{Wa}{N}$$

These maximum power values are represented along the red dashed line on the fan map. On the other hand, the power range is bounded on the lower side with the wind-milling values where almost no work can be transmitted efficiently to the flow. By super-imposing the constraints and range of operation of both the fan and motor maps, we can conceptually perceive the envelope in which both the fan and motor can operate. Of course depending on the relative size between the fan and the motor or the operating

conditions, the envelope will be constrained in different ways. Visually the reader can picture that for a smaller fan and large motor, the fan power lines would be shifted toward the bottom of the motor map and the orange constraints of the motor would be shifted up. Therefore in extreme situations, a large motor operating a relatively little fan would not impose any constraint on the fan operating space. On the other hand, an undersized motor would reduce the fan operating space and will prevent it from operating at high regimes (i.e. high speeds). Using Equation (102) we can parametrically define the power constraint in the pressure ratio/corrected mass flow space.

(104)

$$\pi_{23}\Big|_{Pow_{\max}} = \left(1 + \frac{\eta_{23} \times Pow_{\lim}}{Cp \times Wcs \times Tref} \times \frac{1}{\delta\sqrt{\theta}} \frac{A_{ref}}{A_2}\right)^{\frac{\gamma}{\gamma-1}}$$

Similarly the torque constraint can be formulated as:

(105)

$$\pi_{23}\Big|_{Tor_{\max}} = \left(1 + \frac{\eta_{23} \times Tor_{\lim} \overline{Ncs}}{Cp \times Wcs \times Tref} \times \frac{1}{\delta} \frac{r_{ref} \times A_{ref}}{r_{tip} \times A_2} \frac{2\pi \times Nnom}{60}\right)^{\frac{\gamma}{\gamma-1}}$$

Note: Nnom is the nominal speed and $\overline{Ncs}$ corresponds the corrected and map-scaled speed normalized by Nnom.

The duct imposes two constraints resulting from choking the flow at the either minimum area of the duct (the inlet and nozzle exhaust). Using the flow parameter, a relationship between Mach number mass flow and area was derived from the conservation of mass and isentropic equations:

(106)

$$\frac{Wa}{A_x} = Po_x \sqrt{\frac{\gamma}{To_x \times R}} \left(1 + \frac{\gamma - 1}{2}\right)^{-\frac{\gamma+1}{2(\gamma-1)}}$$

At the inlet, the total temperature and flow are independent from the pressure ratio and efficiency of the fan. Therefore the mass flow constraint from the inlet geometry is

independent from the fan operation and can be represented by a vertical line in the compressor map. The horizontal position of this line in a corrected and scaled map can be determined by introducing the δ, θ, and A_{ref} terms in Equation 106:

$$Wa\big|_{\max} = \frac{Po_a}{P_{ref}} P_{ref} \sqrt{\frac{T_{ref}}{To_i}} \sqrt{\frac{\gamma}{T_{ref} \times R}} \left(1 + \frac{\gamma - 1}{2}\right)^{-\frac{\gamma+1}{2(\gamma-1)}} A_i$$

$$Wc\big|_{\max} = Wa\big|_{\max} \frac{\sqrt{\theta}}{\delta} = P_{ref} \sqrt{\frac{\gamma}{T_{ref} \times R}} \left(1 + \frac{\gamma - 1}{2}\right)^{-\frac{\gamma+1}{2(\gamma-1)}} A_i$$

$$Wc\big|_{\max} = Wa\big|_{\max} \frac{\sqrt{\theta}}{\delta} \frac{A_{ref}}{A_2} = P_{ref} \sqrt{\frac{\gamma}{T_{ref} \times R}} \left(1 + \frac{\gamma - 1}{2}\right)^{-\frac{\gamma+1}{2(\gamma-1)}} A_{ref} \frac{A_i}{A_2}$$

If we assume that γ = 1.4 and R = 286.9 J/kg/K, the equation above can be simplified as:

(107)

$$Wcs\big|_{\max} = 241.303 A_{ref} \frac{A_i}{A_2} \left[kg.m^2 / s\right]$$

The expression for the nozzle constrain is a little more complex to formulate, as the nozzle is located downstream of the fan. As a result, the air total air properties Po_e and To_e are dependent on fan operation.

(108)

$$\frac{Wa}{A_e} = Po_e \sqrt{\frac{\gamma}{To_e \times R}} \left(1 + \frac{\gamma - 1}{2}\right)^{-\frac{\gamma+1}{2(\gamma-1)}}$$

$$\text{with } Po_e = Po_a \frac{Po_2}{Po_a} \frac{Po_3}{Po_2} \frac{Po_e}{Po_3} \text{ and } To_e = To_a \frac{To_2}{To_a} \frac{To_3}{To_2} \frac{To_e}{To_3}$$

Assuming that the inlet compression and nozzle expansion are done reversibly, and we can substitute To_e and Po_e by the following expressions:

$$Po_e = P_{ref} \times \delta \times \pi_{23} \text{ and } To_e = To_{ref} \times \theta \times \left(1 + \frac{\pi_{23}^{\frac{\gamma-1}{\gamma}} - 1}{\eta_{23}}\right)$$

Substituting these formulas back into Equation (108):

$$Wcs\big|_{\max} = Wa\big|_{\max} \frac{\sqrt{\theta}}{\delta} \frac{A_{ref}}{A_2} = P_{ref} \sqrt{\frac{\gamma}{T_{ref} \times R}} \left(1 + \frac{\gamma - 1}{2}\right)^{-\frac{\gamma+1}{2(\gamma-1)}} \frac{\pi_{23}}{\sqrt{1 + \frac{\pi_{23}^{\frac{\gamma-1}{\gamma}} - 1}{\eta_{23}}}} A_{ref} \frac{A_e}{A_2}$$

If we simply this expression by assuming $\gamma = 1.4$ and R = 286.9 J/kg/K, the equation above becomes:

(109)

$$Wcs\big|_{\max} = 241.303 \frac{\pi_{23}}{\sqrt{1 + \frac{\pi_{23}^{1/3.5} - 1}{\eta_{23}}}} A_{ref} \frac{A_e}{A_2} \left[kg.m^2 / s\right]$$

Among the eight inequality constraints in the matching problem only three will vary with flight conditions. These constraints are those related to the motor operation (maximum power –equation (104) -, maximum torque – equation (105) and maximum speed). The other constraints (related to the fan and the duct) can be generalized using the correction and scaling factors described earlier. The fan constraints (surge wind-milling and over-speed) are directly scaled and corrected by the definition of the map. From the manipulation of mass conservation equations and isentropic relationships, it is also possible to correct constraints related to duct choking for fan scale and operating conditions. These corrections and scaling of duct constraints, presented in equations (107) and (109), greatly simplify the identification of the operational envelope of the integrated fan/motor/duct ensemble.

In order to illustrate the generalization presented above and example is provided in this paragraph. This example corresponds to the operational space represented on a constrained fan map. This map is a representation of the E^3 fan (scale 1) at sea level integrated with a 3 MW motor, an inlet throat area and maximum exhaust area of respectively 90% and 60% of the fan face area (note: constraints related to maximum motor torque and speeds are ignored).

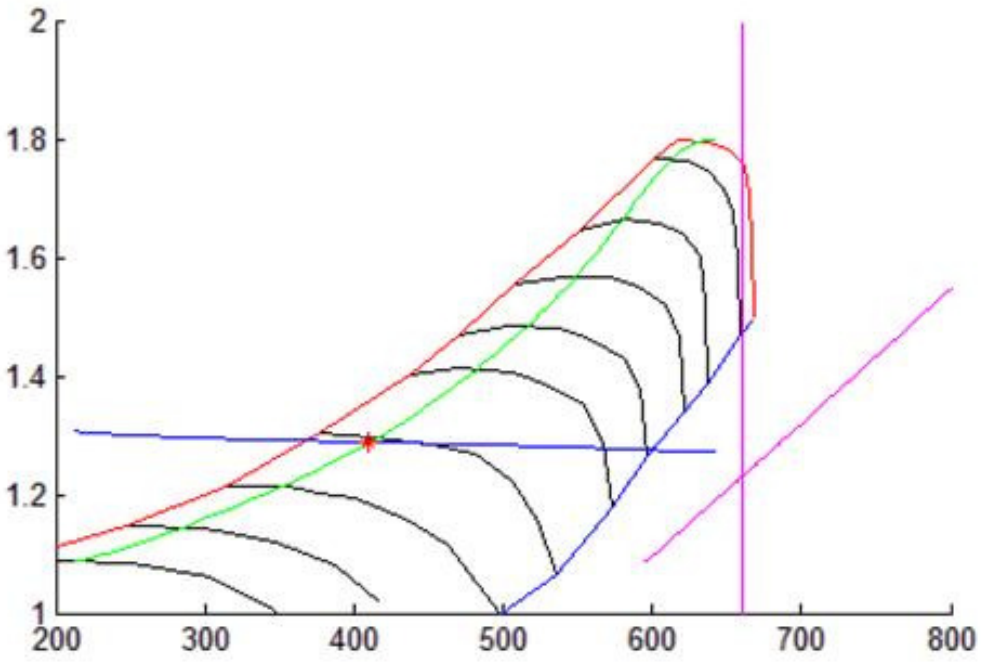


Figure 262: Constrained fan map

Note on Figure 262: For simplicity sake, the fan inefficiencies were ignored in the representation of the motor constraint in this map (i.e. all points between the surge and over speed limits were assumed to have perfect efficiency).

Implementation of the constraint check

As described previously the fan is assumed to run along an operating line. Therefore the space between the surge limit and wind milling limits are reduced down to a line which provides a unique relationship between speed Ncs and the mass flow Wcs. This relationship is given by an external function `Wcst=FanMap(Ncst,Opline,4)`. Based on this assumption, a routine was build to check whether a given speed is feasible, given motor and duct constraints. This routine is named and is reproduced at the end of the appendix. This routine includes a sub-function activated by the variable Pplot. It will plot the operating space using the fan map data and the expression of the constraints (equations (104), (105), (107) and (109)). An example of this output is presented in the figure shown previously (Figure 262).

Two matching algorithms were implemented. The first determines the maximum thrust capacity for all thrust scenarios. The second is used to evaluate the electric power requirement necessary to sustain the thrust level implied by the mission. These two matching algorithms will play distinct roles in the sizing optimization routine in which the performance model will be embedded in. The first algorithm exploring the thrust capacity will be used for testing constraints while the second which is more efficiency oriented will play a role in the objective function of the optimization process.

Maximum thrust capacity evaluation

This routine will identify the maximum thrust that can be produced by a design at given operating conditions (flight M_a and Altitude). Therefore this routine will need to explore the boundary of the operational envelope. Using the concept of operational line, this line will find the highest speed value on the maximum pressure ratio operating line which will comply with all constraints.

Solving for the maximum speed is performed in two steps. First, the maximum corrected-scaled speed (Ncs) allowable by duct' constraints is determined. As highlighted earlier these constraints are valid for any operational condition in the mission. In order to determine this maximum allowable speed a routine was constructed based on the following flowchart:

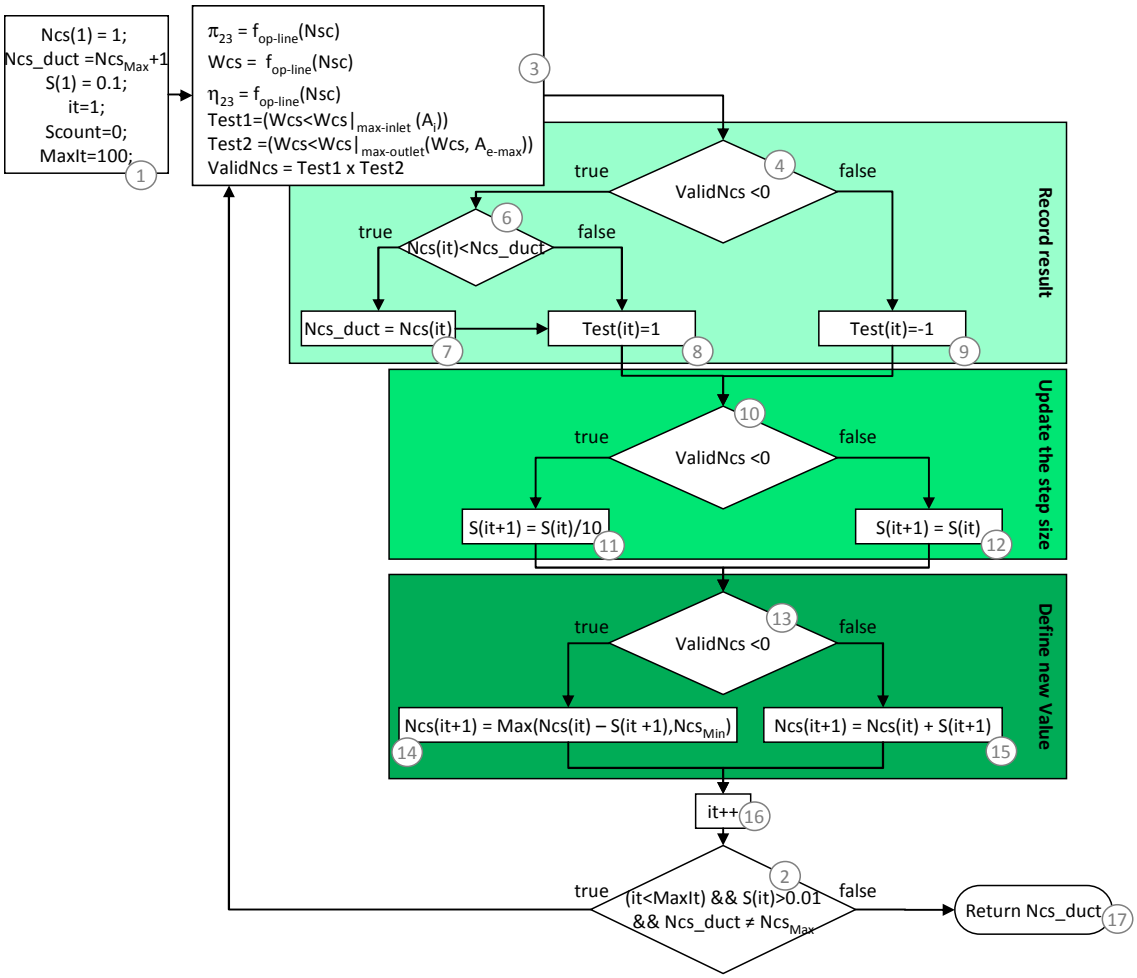


Figure 263: Duct and motor constraints solving

In most situations this routine will exit at its first iteration as one would expect that the duct should not prevent the fan from operating at its maximum speed.

The constraint imposed by the motor limits on the fan is going to change with the operating conditions (δ and θ). Therefore the fan maximum allowable speed must be reconsidered for every scenario. Also unlike the duct constraint which in most situations will not be constraining on operations, the motor constraint is more likely to be active. Therefore in order to avoid relying on an iterative process which significantly slows down the solving process, a regression was used. This regression is based on the following solution of the following equation:

(110)

$$Ncs\_motor / \left\{ \pi_{23}(Ncs_{\max}) = \left(\frac{\eta_{23}(Ncs_{\max}) \times Pow\_cs_{fan}}{Cp \times Wcs(Ncs_{\max})} + 1 \right)^{\frac{\gamma-1}{\gamma}} \right\}$$

with $Pow\_cs = Pow_{\lim} \times \frac{\sqrt{\theta}}{\delta} \frac{A_{ref}}{A_2}$

The form of the regression is presented in the following equation

$$Ncs\_motor = \begin{cases} 0 & \text{if } (Pow\_cs < Pow\_cs_{\min}) \\ f(Pow\_cs) & \text{if } Pow\_cs \in [Pow\_cs_{\min}, Pow\_cs_{\max}] \\ Ncs_{\max} & \text{if } (Pow\_cs > Pow\_cs_{\max}) \end{cases}$$

Hence the maximum thrust is defined by the largest value of Ncs which meets the constraints imposed by the motor and the duct. The code searching for the maximum feasible value of Ncs is the Matlab function `MaxTrustCap3`. Note, the validity variable TestNcs is returned by the function `TestConstraints` presented earlier. Once the maximum feasible Ncs is defined, the maximum thrust capacity is evaluated by the Matlab function `FanModel` (presented at the end of the appendix). `MaxTrustCap3` constitutes the core of the "capability model" presented in the overall breakdown of the electric fan sizing model (see Figure 253 and Figure 254). The "capability model" is embodied by the function `ThrustCapModel` listed at the end of this appendix. Since `MaxTrustCap` can check the scenario one at a time, `ThrustCapModel` is only a wrapper which feeds scenario by scenario the operating conditions to `MaxTrustCap` and defines the constraint witness (noted `Violation`) by checking that the max thrust capability exceeds the max thrust requirement for each scenario. An overview of the process of estimation of the maximum thrust is presented in next figure.

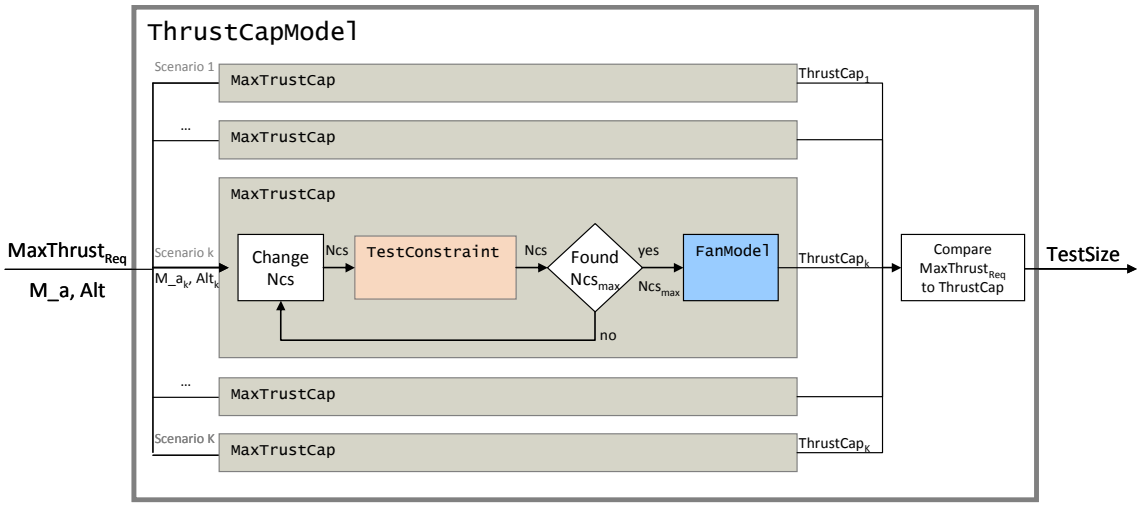


Figure 264: Overview Thrust Capability Model

Once it is confirmed that the maximum thrust requirement can be fulfilled, the sizing routine will determine the energy requirement from the fan (Fan op. model in Figure 253 and Figure 254). The fan regime necessary to provide for the thrust target is defined iteratively. This iterating process starts from the maximum thrust regime. From this maximum regime the speed of the fan $\overline{Ncs}$ is progressively decreased. Following an iterative process similar to the one presented in Figure 263, the `Thrustpoint` function will solve the following equation:

(111)

$$Thrust_{\text{target}} = Thrust\left(\overline{Ncs}\right)$$

The function Thrust($\overline{Ncs}$) is implemented using the `FanModel` function. The iterative process was implemented into function `Thrustpoint`. Once the regime corresponding to the target thrust is identified `FanModel` is used once more to define shaft properties. Function `Thrustpoint` is presented in page 578.

B.3.5 Weight Model

The weight model used in this analysis was derived from a routine written by Mark Waters. This routine was based on WATE [94].This section will not be discussed in much depth as limited transformations from the original model were necessary to integrate Water's model into the current project. The Matlab code is available for the weight module is available page 583.

B.3.6 Elements not captured by the analysis

- Friction in the duct
- Ram drag and heat exchange from the cryocooler heat exchanger
- Weight penalty due to nozzle actuation

B.3.7 Discussion of the integrated performance model

Operation of the engine

Figure 265 presents the efficiency profile of an engine under four typical operating points. These conditions represent take off, initial climb, final climb and cruise.

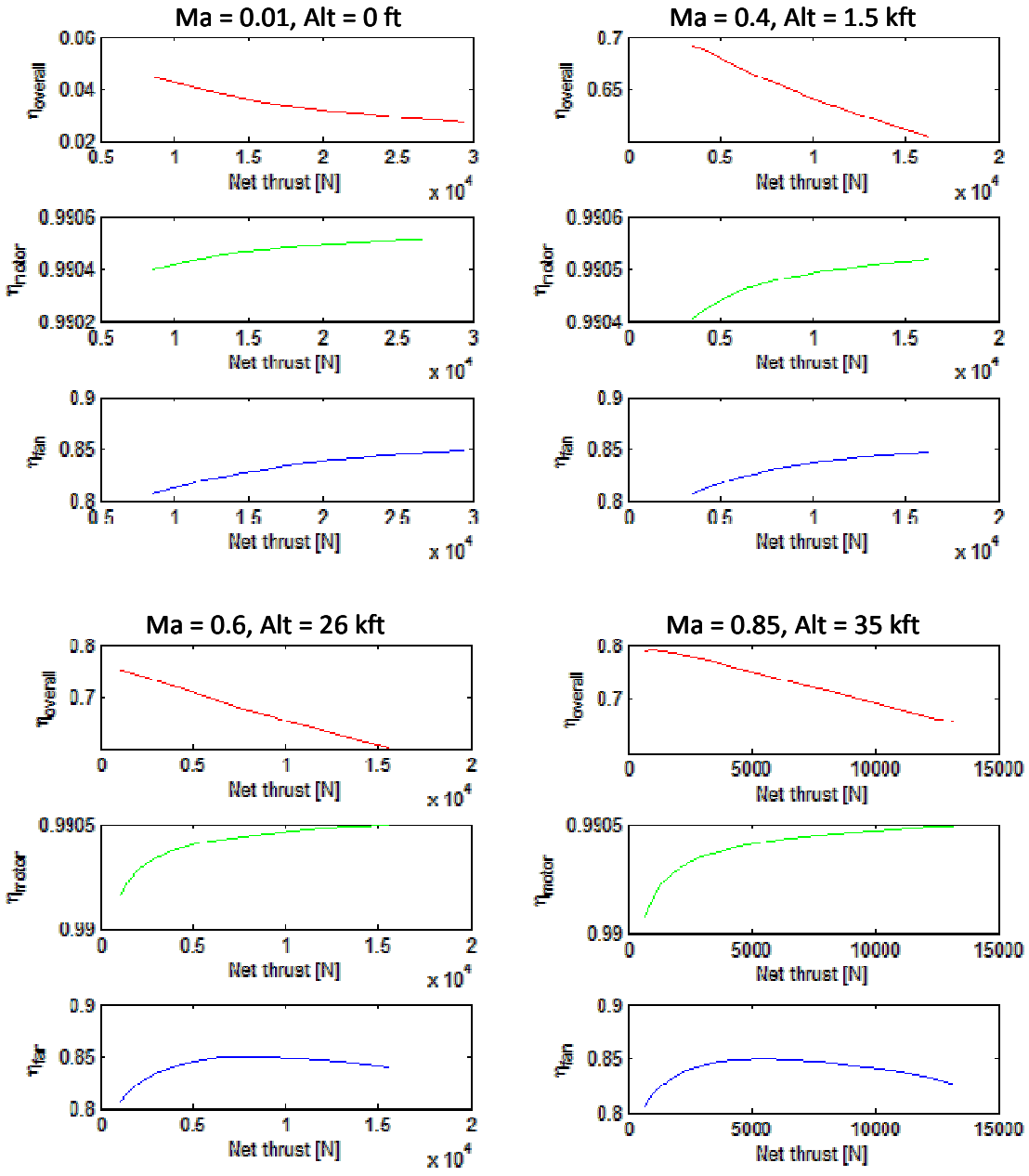


Figure 265: Typical efficiency profile

Engine performance characteristics

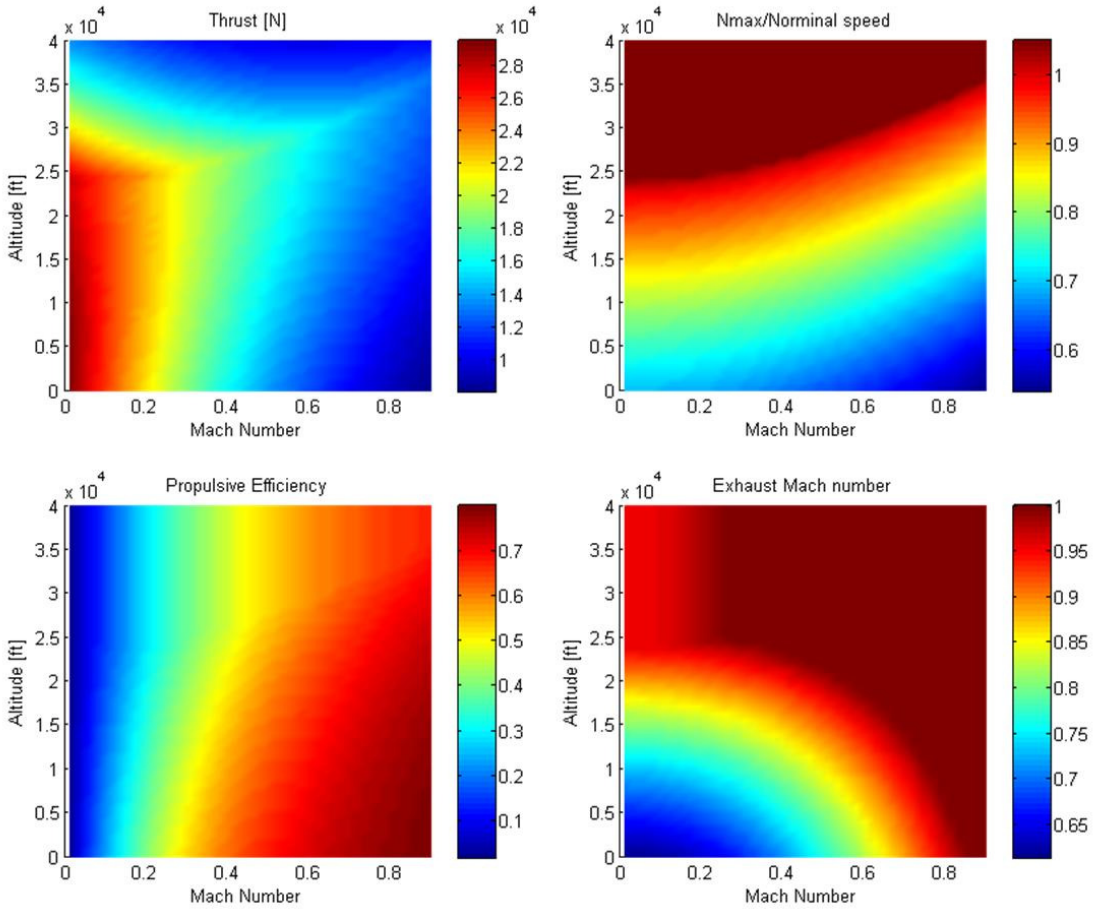


Figure 266: Typical engine characteristics at max thrust

Using the models described earlier, typical characteristics of an electric fan at maximum thrust was observed. Figure 266 presents the characteristics for an engine constituted of a $1m^2$ fan and motor with a 170mm armature radius with a shape factor of 4. Figure 266 shows several operational characteristics at maximum regime. These characteristics are shown for Mach number between 0.01 (quasi-static) and 0.9, and altitude between 0 ft (sea-level) and 40,000 ft.

On these plots, it is interesting to note that the operational space can be divided in several regions. In each region different element of the engine will limit its thrust and will shape its overall efficiency. If we look at the speed plot, we can see that the fan can not operate at full speed at lower altitude. To illustrate this situation, Figure 267 shows a fan

map displaying the duct and motor constraints for operation at Sea Level and 0.08 mach number.

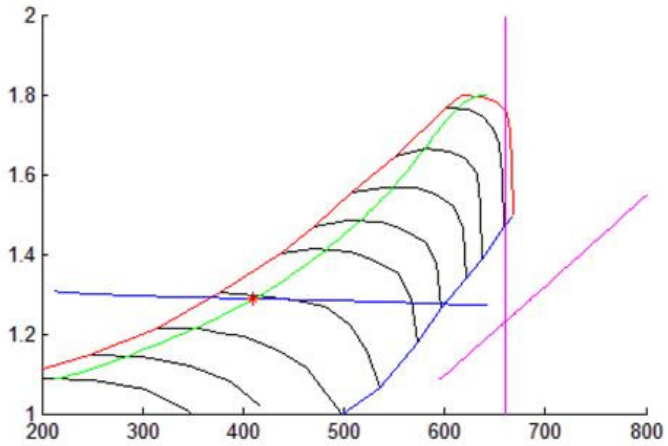


Figure 267: Constrained fan map at low altitude

If we look at this map, we can see that the engine will be constrained by the motor torque limit. As the altitude increase, the constraint associated with the motor power limit will move up in this map (which is based on corrected mass flow and speed). This is due to the fact that, as the altitude increases (or as M_a decreases), δ (pressure correction factor) is dropping. If we refer to equation (105), we can see that as δ decreases the maximum pressure ratio achievable for a given amount of power increases. Therefore at low altitude the maximum thrust achievable is likely to be constrained by the motor. Hence at higher altitudes the motor power or torque limitations are less constraining and the thrust limitation results from the fan speed limit.

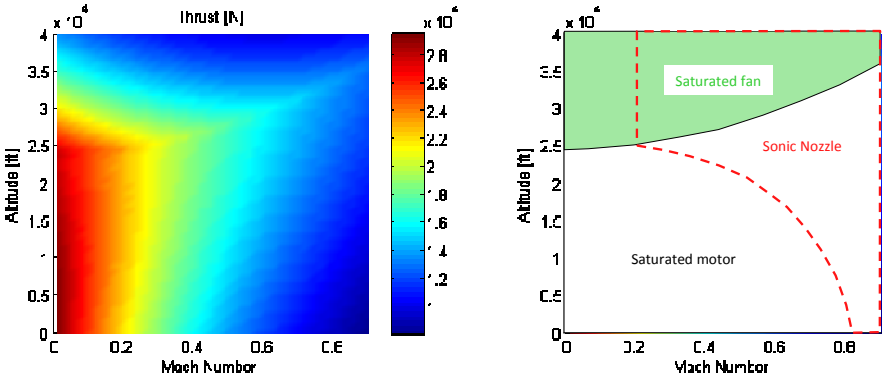


Figure 268: Operational regions

The other discontinuity in the operational space comes from the two possible regimes at the exhaust (fully expanded subsonic or sub-expended sonic). Figure 266 shows the exhaust Mach number. Figure 268 shows side by side the thrust profile of the engine and the different operational zones described earlier. This highlights how the different elements constituting the engine are likely to be constrained on their sizing. The motor is most likely to be constrained during take-off and initial climb while the fan will most likely be sized at high altitude operation (final climb or cruise).

We can also not that as the motor is growing relatively larger with respect to the fan, the saturated fan zone will increase along with the sonic nozzle region (more energy injected in the flow).

General topology of the maximum thrust attribute

The following figure presents the topology of the thrust capability for take off conditions. One axis presents the design variables ro which scales the motor while the other presents the fan area design variable which scales the duct and the fan. These two design variables are the major contributing factors on the thrust capability responses.

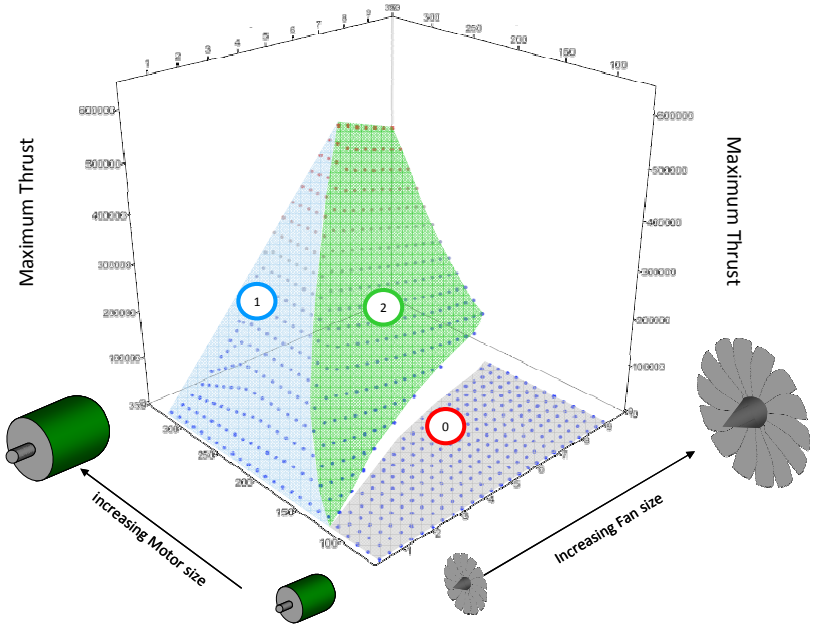


Figure 269: Topology of the thrust capability

The design space can be broken down into three zones. The zone noted as (1), corresponds to designs which are constrained in their thrust by the size of their fan. We can see that, within this zone, the size of the motor (ro) does no influence the max thrust capability. Zone (2) corresponds to the engine designs where the maximum thrust is primarily constrained by the size of the motor. It is important however to note that unlike zone (1) which was insensitive to one of the design variables (ro), zone (2) is influenced by both the fan size and the motor size. Indeed we can see that the the maximum thrust that can be produced by a given motor size increases as the fan area increases. The flat zone (0) represents engine designs incapable of producing any thrust. They are not operational because their motors are too small with respect to the fan.

B.4 Engine sizing

The engine is sized based on the following optimization problem:

(112)

$$\underset{\overline{X}}{Minimize} \quad OEC\left(\overline{X}\right) = \gamma_1 \frac{m_{engine}\left(\overline{X}\right)}{m_{engine-baseline}} + \gamma_2 \frac{diameter\left(\overline{X}\right)}{diameter_{baseline}} + \gamma_3 \frac{1-\eta_{CZ}\left(\overline{X}\right)}{1-\eta_{CZ-baseline}} + \gamma_4 \frac{PowEreq\left(\overline{X}\right)\Big|_{\max}}{PowEreq\Big|_{\max-baseline}}$$

$$\text{S.t. ThrustMaxReq}\left(\overline{X}\right)_k - \text{ThrustMaxReq}_k < 0 \text{ for all } k \in [1, K]$$

With

X(1): Afan – Fan face area [m^2]

X(2): ro – Mean armature radius [mm]

X(3): Arm – Machine shape factor [La/ro]

Given the number of constraints, this optimization problem is relatively complex to resolve. On the other hand, it is essential that an optimum is found promptly (as this model will be used within a multi-level optimization loop).

In the development phase of the model, several optimization approaches were tested (gradient based, Sequential Quadratic Programming – SQP – or genetic

algorithms). Both the gradient and SQP approaches which were used in conjunction with a penalty function to include the constraints were un-robust with respect to their starting point. Hence if the starting point was unfeasible they would not be able to identify feasible space. But even if the starting point was feasible, they would both have significant difficulties identifying the global optimum. This lack of robustness can partly be explained by the discussion in the previous section. Indeed the capability is limited either by the fan and the motor. Therefore the relationships between the design variables and the capability (which is an element of both the objective and the constraints) is non linear. Therefore the most robust solution was the genetic algorithm which identifies better optimum values. But genetic algorithm is not a practical solution because of its slow convergence.

In order to have both a practical and robust optimization process, it was therefore impossible bluntly place a black box optimizer on the problem while hopping for a prompt and robust sizing of the engine. Since there is little that can be done to accelerate the convergence of stochastic algorithms, the genetic algorithm approach was not investigated further. Therefore efforts were dedicated to the improvement of the robustness of SQP optimization algorithm. The successful implementation of the SQP was made possible via the implementation of three solutions:

- Initiation of the optimization with feasible initial guess
- Use of a penalty function to unsure that the search remains within the feasible space
- Using Neural Networks (NN) as a substitute to the physics-based model to accelerate the optimization process

Each of these techniques provided essential advantages. These advantages will be described in the following paragraphs.

B.4.1 Pre-sizing

In order to make sure that the optimizer is robust in finding both a feasible and reasonably optimal design, a pre-sizing routine was used to provide an educated guess to the optimizer. This pre-sizing process is composed of two successive steps:

- The fan pre-sizing analysis
- The motor pre-sizing analysis

Fan pre-sizing

The fan pre-sizing analysis will size the fan and duct, assuming that the motor driving the fan is infinitely powerful (i.e. assuming the motor has no power or torque limits). In the engine analysis, each scenario imposes a constraint. Figure 270 presents a notional constraint in the ro A_{fan} space.

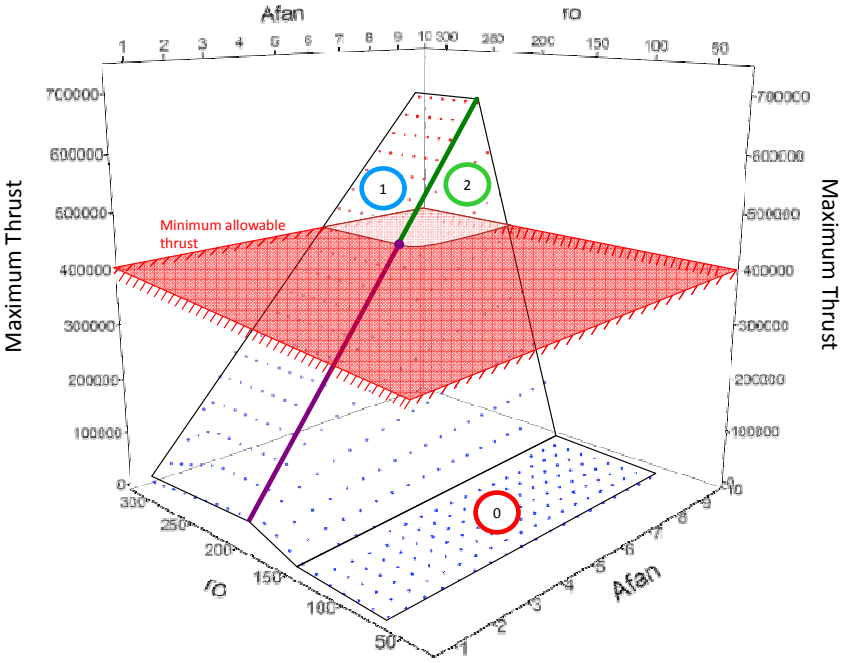


Figure 270: Constraint in the ro-Afan space

Based on the analysis presented earlier, we know that once the motor size exceeds some critical size, the thrust that can be achieved by the engine is only a function of the fan size. If we project the surface corresponding to zone 1 in the ro-Afan-Max_thrust

space into the Afan-Max_thrust space, the thrust constraints can be translated into a minimum fan area (see Figure 271).

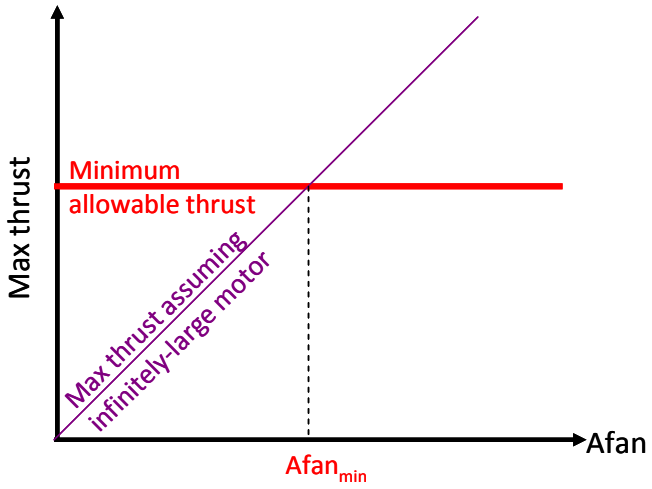


Figure 271: Notional fan area constraint

The engine is sized for an ensemble of scenarios. Each scenario brings a different constraint. These constraints will differ in required thrust (max thrust at take-off larger than climb or cruise) but also in the slope of the purple curve which accounts for altitude and Mach effects (for a given a larger fan area is necessary as speed and altitude increase). A notional example is presented below:

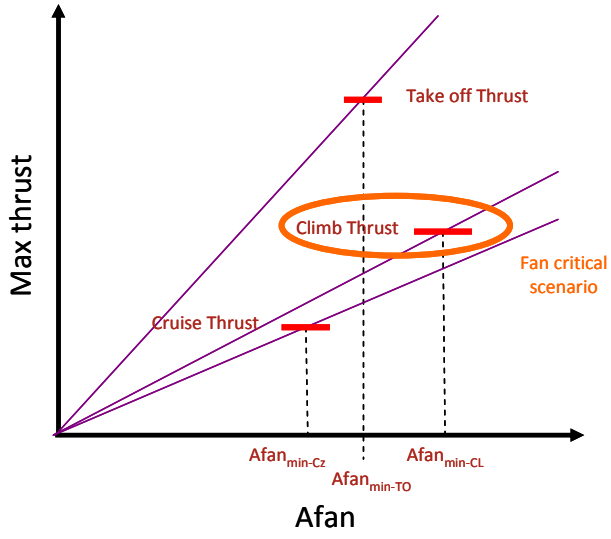


Figure 272: Identification of the critical sizing scenario

Therefore the fan will be pre-sized by looking at each scenario in series, and by setting the pre-sized fan area as the most constraining fan area. Note: The scenario which requires the largest fan area is the sizing critical scenario.

Motor pre-sizing

Using the pre-sized fan area, a solver will evaluate the shaft characteristics if the pre-sized fan was to provide precisely the maximum thrust required by the mission. These shaft characteristics allow for the identification of the maximum power, torque and speed required from the motor. The pre-sized motor will be the smallest motor which operation limits (Pow_{lim}, Tor_{lim}, and N_{lim}) will exceed those required by the pre-sized fan. It is to be noted that the operational limits of the motors are a function of all Ks, ARm and ro. In order to simplify the pre-sizing process a pre-defined value is used for ARm and Ks. Therefore the motor is pre-sized by solving for the smallest ro capable of supporting the pre-sized fan.

Overview of the pre-sizing procedure

In order to summarize the procedure presented in previous paragraph and to provide an overview of the Matlab routine created to pre-size the engine, a flow chart of the pre-sizing procedure is presented below.

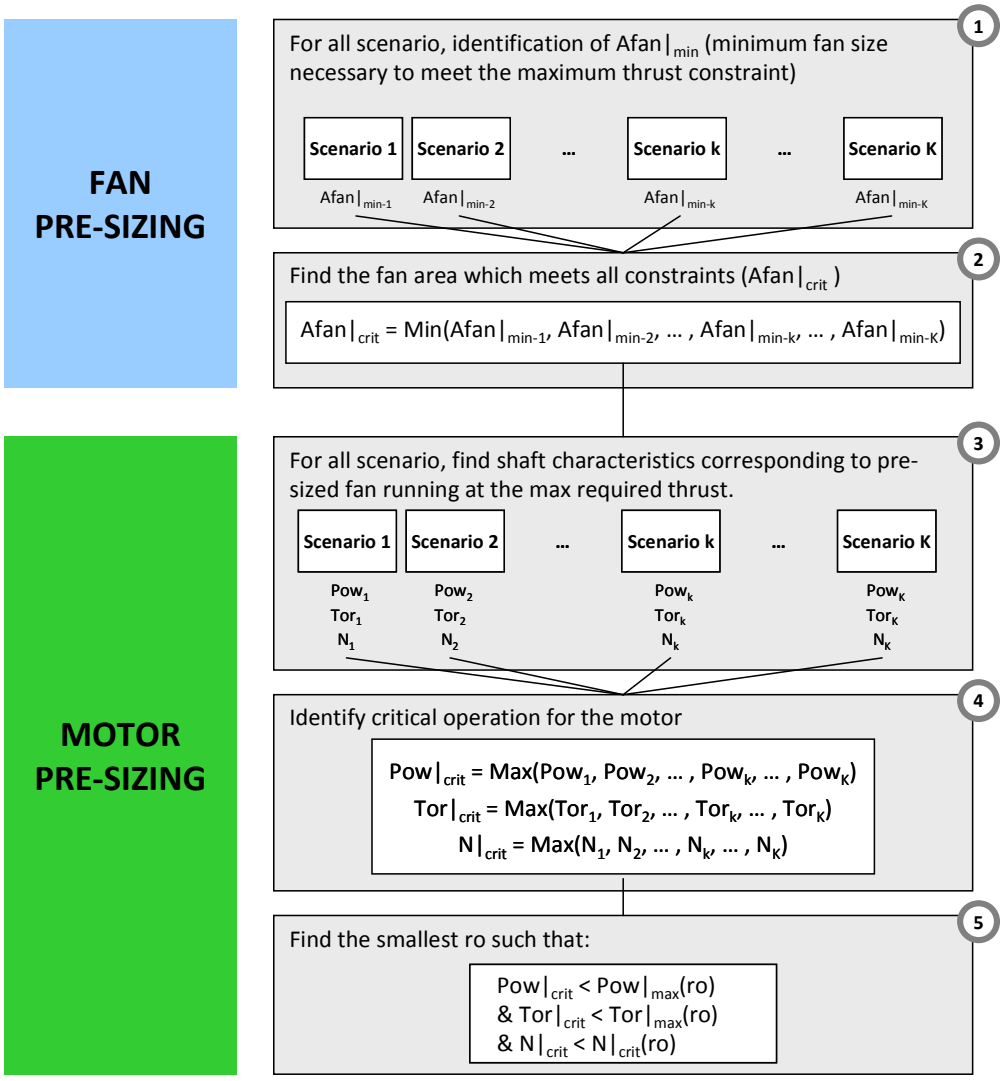


Figure 273: Engine pre-sizing process

The routine used to pre-size the engine are FanPreSizing and MotorPreSizing. These routine are presented page 588 and 589.

B.4.2 Penalty function

The OEC presented in equation (112) includes several terms that cannot be computed is the engine is infeasible. For example, it is not possible to estimate the efficiency of an engine at a thrust regime it can not produce anyway. There the objective function is undefined for any unfeasible value. To resolve issue two options were available. The first is to provide a dummy OEC value when the design in unfeasible. The

second is to make sure that the search optimizer starts and remains in the feasible design space. The solution implemented was a conjunction of both options.

In a first step, a value was assigned to the objective function for unfeasible design. For practical purposes the value assigned was a constant. Other strategies like linear exterior penalty functions[95] were unsuccessfully attempted (generation of instabilities along constraint boundaries which would attract the optimization path on feasible regions near the boundary). The constant value assigned to the infeasible region was 10. This value was chosen because it is one order of magnitude over the feasible objective function (OEC~1). This value was sufficiently high to prevent the search algorithm to venture outside of feasible space. Choosing higher value was counter productive as it would cause the optimization process to be halted prematurely. An explanation of this premature halt was probably caused by the observation of gradients exceeding some optimization settings.

But alone this resolution was not sufficient and could only be used to prevent the optimizer from "escaping" the feasible space. If the optimizer was to leaving the feasible space it would not be able to effectively progress back to feasible space as the objective is uniform in infeasible region.

(113)

$$P\left(\overline{X}\right)=\frac{rp}{\max\left(ThmaxReq\text{-}ThCap\left(X\right)\right)}$$

In order to avoid this problem an interior penalty function was implemented. The penalty function is presented in equation. This penalty asymptotically approaches infinity as the search approaches the boundary

(114)

$$F\left(\overline{X}\right)=\begin{cases} OEC\left(\overline{X}\right)+P\left(\overline{X}\right) & \text{for } \overline{X} \text{ feasible} \\ OEC_{\inf} & \text{for } \overline{X} \text{ infeasible} \end{cases}$$

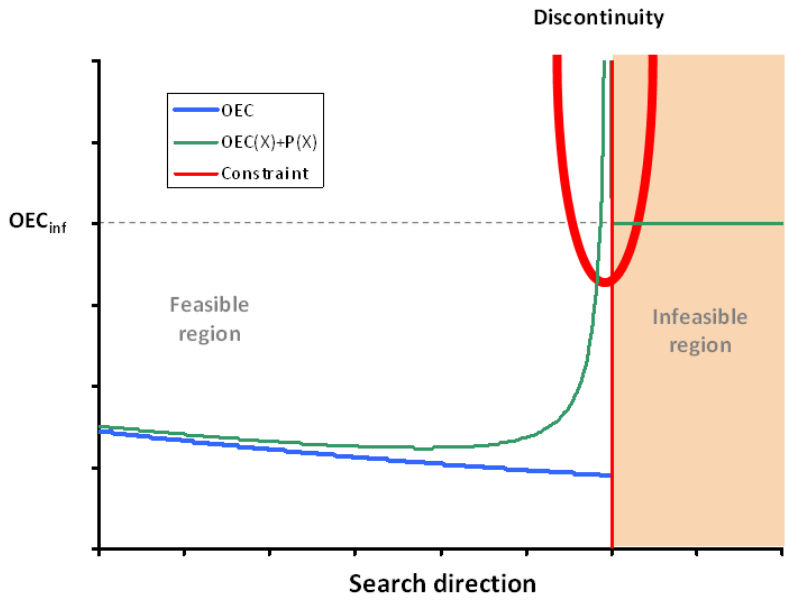


Figure 274: Penalty function discontinuity

But alone this penalty formulation was not sufficient to protect the optimization process as it creates a discontinuity near the boundary on the feasible side (see Figure 274). This discontinuity would throw the optimizer outside of the feasible space or prevent its return in the feasible space. Therefore an additional condition was used to remove this discontinuity:

(115)

$$F(\overline{X}) = \begin{cases} \min\left[OEC(\overline{X}) + \dfrac{rp}{\max(ThmaxReq\text{-}ThCap(X))}, OEC_{\inf} \right] & \text{for } \overline{X} \text{ feasible} \\ OEC_{\inf} & \text{for } \overline{X} \text{ infeasible} \end{cases}$$

This penalty formulation has proven to be sufficiently robust to provide a stable sizing process.

B.4.3 Surrogate modeling of the ducted electric fan model

To accelerate the optimization process, the relationships defined by the physics-based model had to be captured by a set of surrogate equations. These relationships included the thrust capability, the weight and the efficiency of the engine. Since these

regressions were to be used in an optimization context, several aspects were considered in defining the surrogate model:

- The accuracy of predicted values
- The absence of ripple which could possibly trap the optimizer into a local optimum
- Ability to capture both continuous values and discrete values (like zero thrust which implies that the engine is dysfunctional)
- Avoid the creation of discontinuity in first order derivatives with no physical meaning

The responses were fitted based on a sample of data. This data was generated using a full factorial Design of Experiments (DoE) with 25 levels on all design variables (Afan, ro, and ARm). Based on this sampling data, the design space was broken down in the three zones presented earlier in Figure 269 and reproduced in Figure 275.

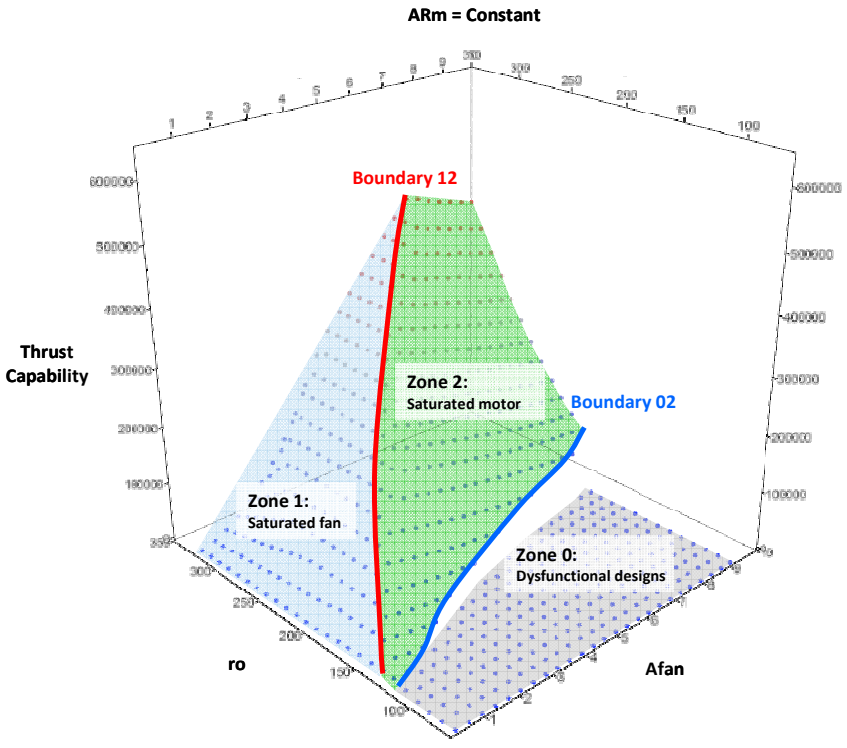


Figure 275: Fragmentation of design space

For each zone surrogate methods were individually deployed so that the response shape specific to the zone could be accurately captured. Based on the surrogates corresponding to each zone, a joining algorithm was deployed to avoid unrealistic discontinuity susceptible to perturb the optimizer search. The following paragraphs will explain why and how surrogates models were generated as described. The first paragraph will explain the motivation behind braking down the design space into zones. The second paragraph will describe the technique used to split up the design space. The third paragraph will review the individual responses (thrust weight and efficiency) captured by surrogate models.

Fitting approach

As discussed previously, the behavior of the electric fan engine is conditioned by several competing physical limitations. Some of these limitations include the maximum fan speed or the torque or power limitation of the motor. Each of these limitations act as internal constraint which define the relationships between the design variables (fan area, motor radius, etc…). Then depending on the design of the engine, different limitation may be active, and different relationships are active. An illustration of these changing relationships was clearly highlighted in Figure 269. On that plot, we can see that in zone 1 a first relationship exist between the design variables (Afan, ro, ARm) and the response (Thrust capability (ThCap). For complex and changing relationships as the one highlighted above it is difficult to obtain an accurate surrogate model which will capture all relationships. To illustrate this challenge a direct neural net was fitted to the entire design space for the thrust capability sea level static conditions. The predicted response is represented along with the training data is represented in. On this figure we can see that the discontinuity in response shape and derivatives are very difficult to capture with a single continuous function. But beyond the fact that the response can not be accurately captured, this difficulty has an important negative consequence which is to artificially

create “ripples” near the discontinuities. These ripples are unacceptable as they would artificially generate local optima or disjoint feasible space with no physical significance.

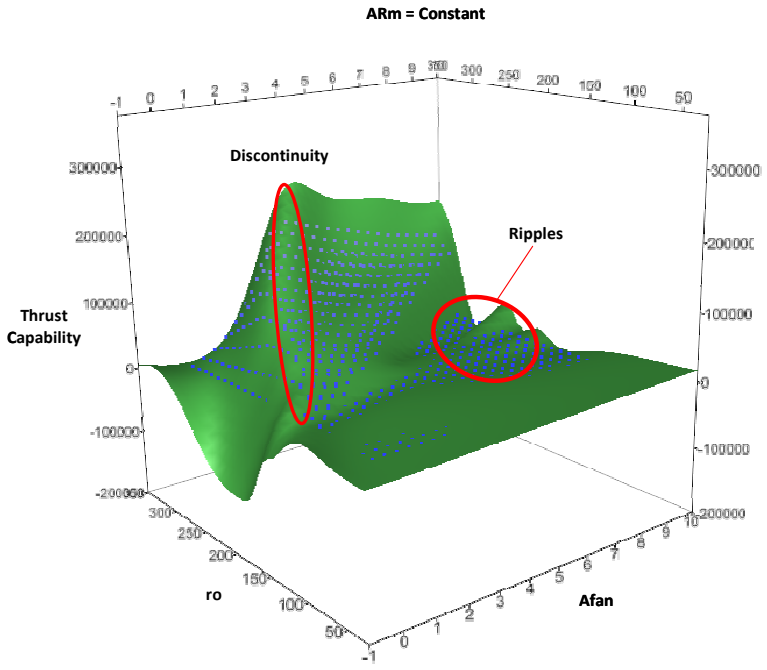


Figure 276: Issues with single fitting

In order to avoid both the lack of accuracy and the generation of ripple in the response, different surrogate models were created for each of the zones depicted in Figure 275. The overall design space surrogate model was created by the logical junction of the surrogates of the three zones.

The segmentation of the design space requires the definition of the coordinates of boundaries 02 and 12. Based on their coordinates one can classify the regions of the design space to zones 0, 1 and 2. Therefore we will first describe how the boundary coordinates were defined. Then we will discuss how the junction between the predictions for each zones were defined in a fashion which avoided discontinuity.

The coordinates of the boundaries were defined by monitoring the thrust capability response. In each zone the thrust response has a particular behavior which can be detected by a simple filter. In zone 0 (Dysfunctional designs), the thrust capability is equal to zeros. In zone 1 the thrust capability is insensitive to chances in ro. For each

Afan/ARm combination, it possible to detect the position of the boundaries by scanning data along ro. For small ro values, the thrust value is 0. If we increase the value of ro, boundary 02 is located at the point where the thrust capability goes from zero to some non-zero thrust value. If we continue increasing ro, the thrust capability will stop increasing. This implies that we have localized boundary 12.

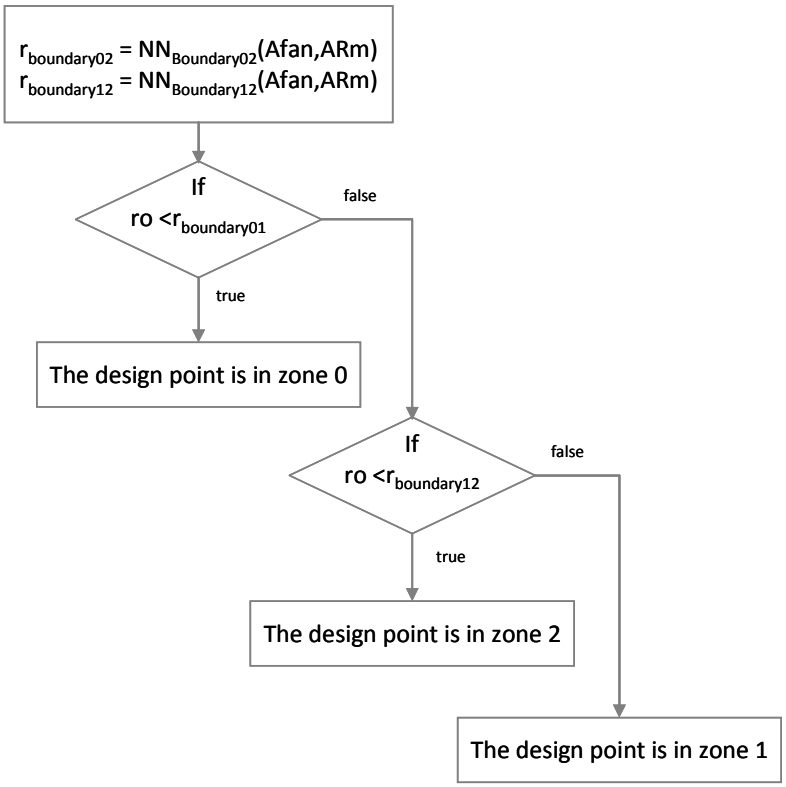
Using this scanning method for each combination of Afan/ARm the approximate position of the boundaries can be defined by the set of coordinates (Afan,ro,ARm) where the discontinuities occur. This set of coordinates is used to train a set of neural nets (NN). These nets allow defining the following function:

(116)

$$r_{boundary} = NN\left(A_{fam}, ARm\right)$$

Boundary 02 indicates the limit where the fan is dysfunctional; therefore this limit is independent of the flight scenario. But the boundary 12 (separating designs with max thrust capability limited by the fan from those limited by the motor) is dependent on the flight altitude and mach number. Therefore boundary 12 is captured by a different set of nets for each flight condition. In the mission considered in the thesis 8 flight conditions are used to size the engine. Therefore 8 sets for nets are defined to characterize boundary 12.

Once the coordinates of the zones are identified, the design points constituting the DoE are classified by zone. This classification is done for each flight condition. The logic ruling the classification of the design (Afan, ro, ARm) is as follows:



The data classified under zones 1 and 2 will be used to train a regression representing the responses specific to their zones.

The training of the neural nets was performed using the Matlab neural net training toolbox. The interface to the toolbox was provided by a tool developed by Carl Johnson called BRAINN (Basic Regression Analysis for Integrated Neural Networks).

Definition of surrogates

Different responses have different behaviors and were represented in different ways. The following paragraphs are used to describe the specificities of each response.

The thrust capability (noted ThCap), is defined by the maximum thrust that the engine can produce. The topology of this response was shown in Figure 270 (reproduced in Figure 275). For simplicity sake the response was regressed for each flight conditions and the following discussion on regressing ThCap assumes that the flight conditions are fixed.

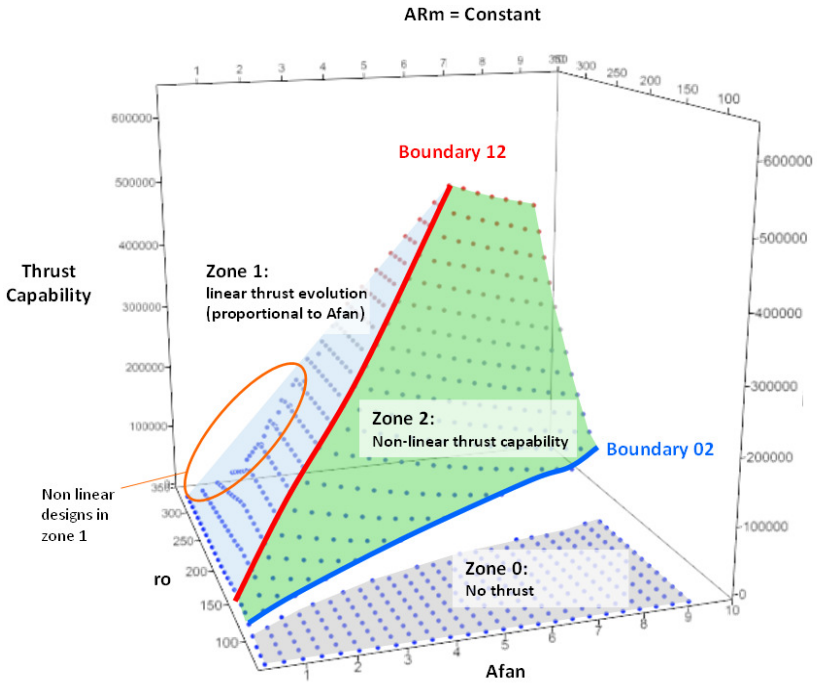


Figure 277: Thrust capability evolution by zones

Figure 277 presents the thrust capability behavior as a function of Afan and ro. On this plot we can see that in zone 1 the thrust capability evolves proportionally with Afan. It is also important to note that the proportionality factor between Afan and thrust is insensitive to the ARm (i.e. it does not change with the shape of the motor). A notable exception can be observed near the high ro-value/ low Afan-value corner. This exception results from the fact that the rotation speeds of smaller fan are proportionally higher than

larger fan. On the other hand, as the motors becomes larger, the loads on the windings are increased which tends to impose lower speed limits on the motor. The conjuction of these two effects cause this singularity highlighted in Figure 277. But ignoring this singularity is not expected as problem because these types of engines are clearly suboptimal from many points of view (the over-sized motor implies a weight penalty and the undersized fan implies an efficiency penalty). Given the obvious sub-optimality of this corner of the design space, no special efforts were dedicated to capture this singularity.

Therefore in order to capture the response in zone 1, the area-specific thrust was estimated for each point on boundary 12. The median value of the area-specific thrust was used to characterize the slope of the thrust response in zone 1. Most of the area-specific thrust measurements were identical, therefore the median value was chosen instead of the mean in order to avoid polluting the estimate by the potential presence of a singularity in data to bias the estimate. The response in zone 2 has a non-linear behavior with no discontinuity. This behavior can be very accurately captured by a 10 layer neural nets. For points in zone 0, their thrust capacity is defaulted to zero (no regression necessary).

The thrust capability has a continuous behavior across zone 1 and 2. The only discontinuity present in the model is a boundary 02 (going from the strictly positive value of thrust to the zeros thrust capability in zone 0). At boundary 12 there is no discontinuity in value but there is a discontinuity in the derivatives. In order to avoid any discontinuity resulting from the unavoidable fit error in zone 1 and 2, a “transition” band was used inside zone 2 near boundary 12. This discontinuity problem and its resolution are illustrated by Figure 278.

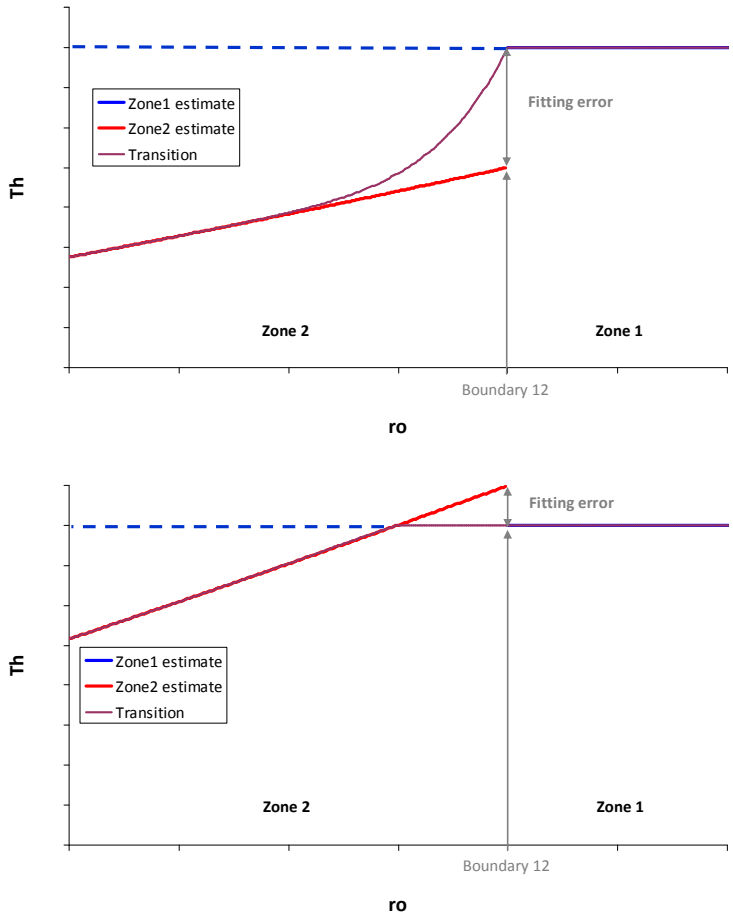


Figure 278: Boundary fitting error

The formula corresponding to this transition was:

(117)

$$ThCap(Afan, ro, ARm) = Min\left[\begin{array}{c}\alpha \times Afan \\ \beta(ro)\times\alpha\times Afan + (1-\beta(ro))\times NN(Afan, roARm)\end{array}\right]$$

$$\text{With} \quad \beta(ro) = \begin{cases} 0 & \text{for} \quad ro < r_{trans} \\ \left(\dfrac{ro - r_{trans}}{r_{boundary12} - r_{trans}}\right)^{p} & \text{for } ro \in [r_{boundary12}, r_{trans}] \\ 1 & \text{for} \quad ro > r_{boundary12} \end{cases}$$

and $r_{trans} = r_{boundary12} - \%r \times (r_{boundary12} - r_{boundary02})$ where p = 5 and %r =40%

This transition prevents the occurrence of discontinuity in ThCap. It also prevents the occurrence of a local maximum ThCap by ceiling the value of ThCap to the zone 1 estimate (largest thrust capacity possible for a given fan area). Using a hyperbolic

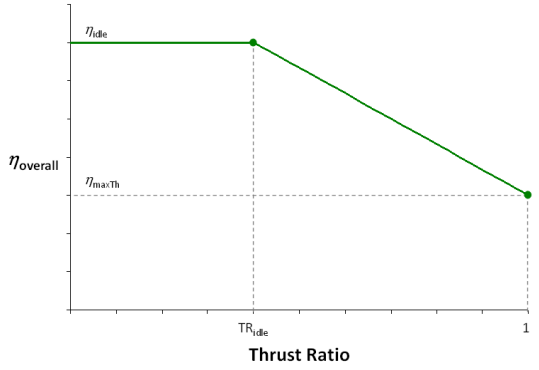
transition (factor p) allows avoiding discontinuity in the first 4 derivatives susceptible to perturb the optimizer. The discontinuity is then only located near boundary 12. This local discontinuity is acceptable as it correspond to a transition between physical limitations.

Based on the observations made in Figure 265, we can see that the efficiency of the engine has almost a linear behavior with thrust. Therefore, the representation of the efficiency was defined as the follows:

(118)

$$\eta(TR, Afan, ro, ARm) = \begin{cases} \eta_{idle} & \text{for} \quad TR < TR_{idle} \\ \eta_{idle} + \dfrac{\eta_{\max Th} - \eta_{idle}}{1 - TR_{idle}} \times (TR - TR_{idle}) & \text{for} \quad TR \in [TR_{idle}, 1] \end{cases}$$

$$\text{with } TR = \frac{Th_{req}}{ThCap} \text{ (Thrust ratio)}$$



In equation (118), three new terms are introduced (η_{idle}: idle regime efficiency, $\eta_{\max Th}$: efficiency at max thrust, and TR_{idle}: Idle thrust to max thrust ratio). It was observed that the primary driver for changes in efficiency was the fan regime. This is due to the fact that the efficiency change with respect to regime of the motor is negligible with respect to the change in efficiency due to fan and propulsive efficiencies. Since idle regime is defined for the fan operating at 0.4 Nc, the idle efficiency (η_{idle}) was observed to be quasi-constant for all design points (only dependent on operating conditions – Mach number and altitude). In a similar fashion all design in zone 1 would have similar max-

thrust efficiencies ($\eta_{\max Th}$). To illustrate these trends, the responses η_{idle} and $\eta_{\max Th}$ are represented over a slice of the design space:

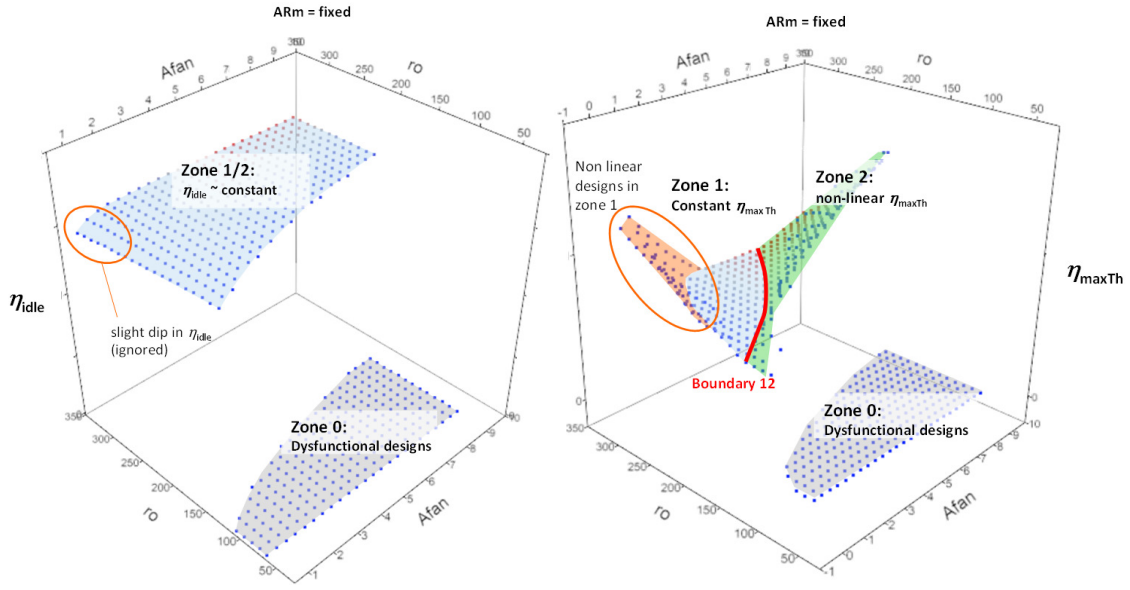


Figure 279: Typical behavior of efficiencies at idle and max thrust

A set of NN were fitted to capture the variations of $\eta_{\max Th}$ in zone 2. The variations of $\eta_{\max Th}$ in zone 1 were assumed to be negligible. This assumption is perceived as being valid since the designs highlighted in orange were deemed suboptimal for the reasons described in previous paragraph.

The regression on mass was significantly simpler than thrust capability and efficiency as its behavior is monotonous and includes no discontinuity except on boundary 02. This discontinuity results from the assignment of zero weight to dysfunctional engines. The typical behavior of the engine total mass is provided in the following figure.

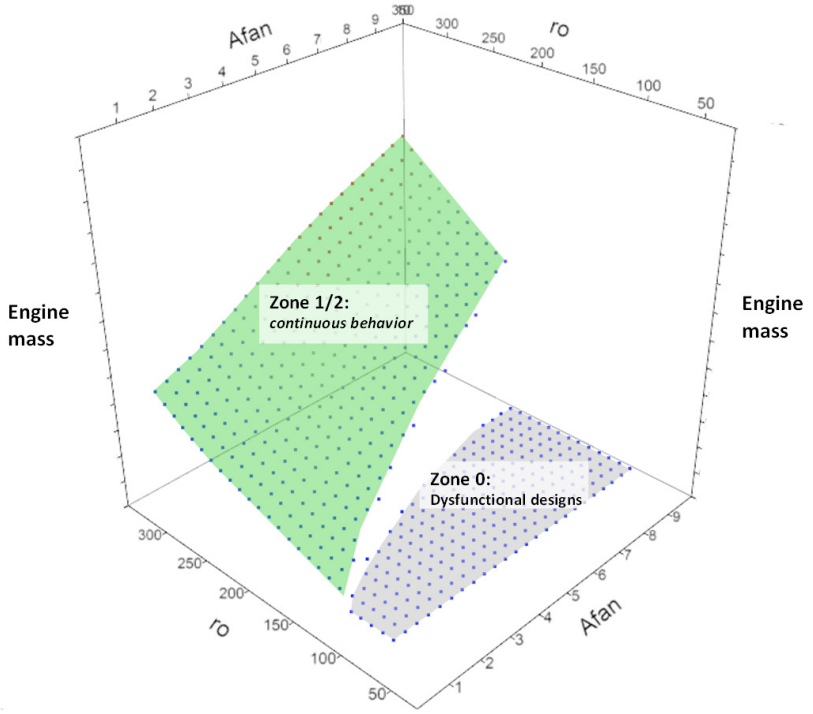


Figure 280: Typical behavior of engine mass

Automated generation of surrogate models

The logic described in previous paragraphs was captured in set of Matlab scripts. The following graphs provides and overview of the process and presents the different elements of the scripts automating the generation of surrogates. The first script is `RunDoE.m` which will setup a full factorial experiment (by default 25 levels on each design variable). Based on this DoE, it will call the function `DuctedFanResponse` which runs the electric fan descriptive model for each experiment. `RunDoE` and `DuctedFanResponse` are presented in pages 590 and 592.

Based on the results of the experiment the data is organized and analyzed by the `FilterAndTrain.m` script. This script organizes the DoE information in a format which can be easily parsed. It then detects the position of the zone boundaries. This allows classifying the DoE data into training sets for the neural nets. The training is performed via the function `Netfit` which interfaces with BRAINN. Script `FilterAndTrain.m` and `Netfit` function are presented in page 593 and 598.

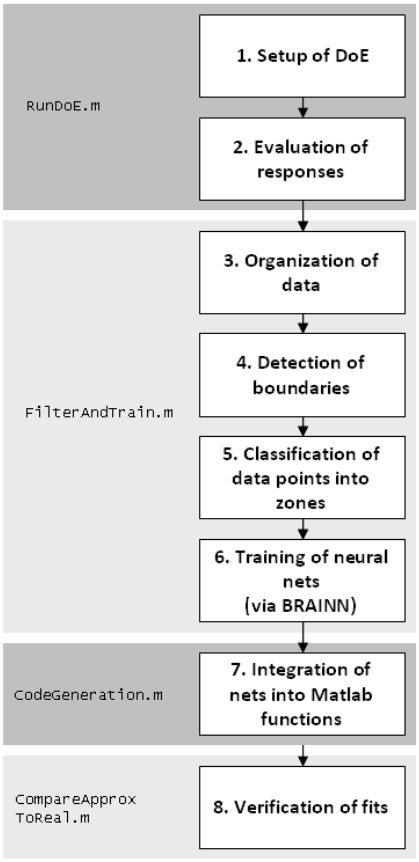


Figure 281: Surrogate model building process

The equations corresponding to neural nets and other regression defined in `FilterAndTrain` are passed in the form of a cell matrix of strings. This matrix is then used by the `CodeGeneration.m` script. This script generates automatically matlab code based on the logic described in the earlier paragraphs. In this code, the equations from the neural nets are integrated. The `CodeGeneration.m` script is presented in page 600. This script generates three regressed models for the engine weight (`Approxm_engine`), for the thrust capacity (`ApproxThrustCap`) and a third for efficiency (`ApproxEff`). Samples for these automatically generated files are provided in pages 604, 605 and 608 respectively.

To verify the quality of the resulting surrogate model, the `CompareApproxToReal.m` script was build. This script runs the automatically generated surrogate functions and compares its result to the actual responses. This script plots the responses over a slice of

the design space (fixed ARm). These plots are presented in the following paragraphs. The code of `CompareApproxToReal.m` is presented in page 615.

Verification of fits

By subdividing the design space, a high quality fit could be achived while based on relatively simple neural nets. The following figure (produced by the ML script) presents an predicted responses from the regression over a slice of the design space for ARm = 2.67 and at flight conditions corresponding to a high altitude climb (FL 275 and M_a = 0.75). The error is represented by green and red bars (green if the error term is negative, red if it is positive).

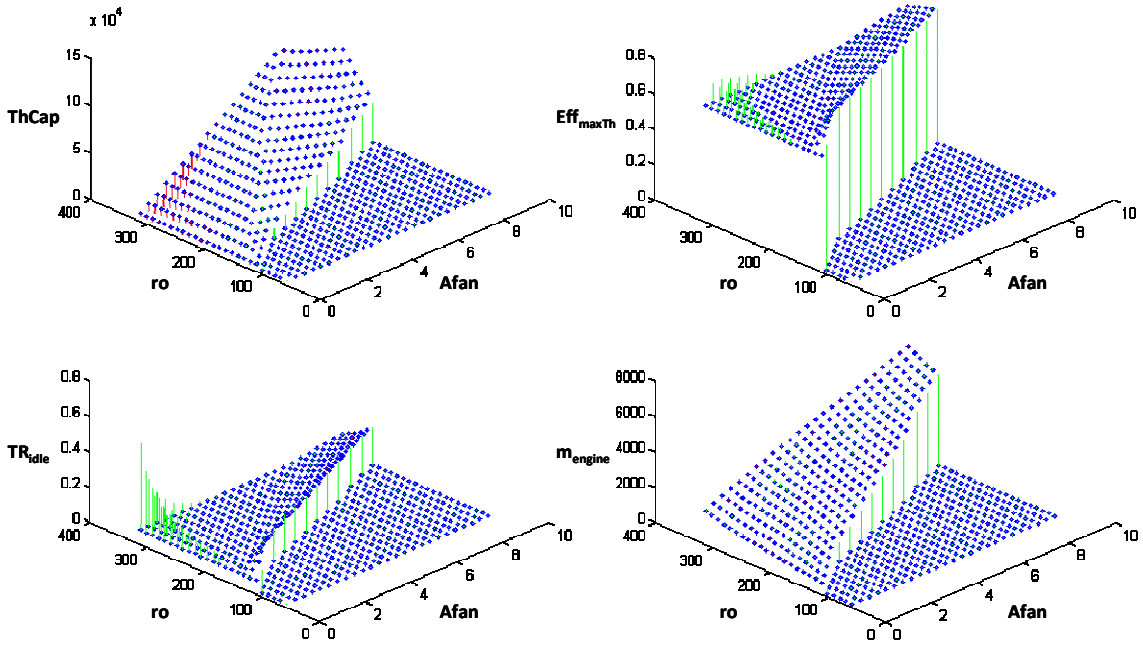


Figure 282: Predicted responses with error representation

This figure gives us a qualitative appreciation of the quality of the fit provided by the appproach presented earlier. First of all we can see that the amplitude of the error is barely perceptible across most of the design space. The behavior of the response in the high ro/ low Afan region was not captured. But based on the justifications presented

earlier, claiming that this corner of the design space is suboptimal, this error was considered as having no insidence on the applicability of the model.

The other source of uncertainty comes from the boundary between zone 0 (dysfunctional designs) and zone 2 (designs constrained by the motor). This error resuls from slight inacurracies in the prediction of the coordinates of the boundary (reminder the coordinates of the boundary is cauptured by a NN). But it is important to understand that the large error amplitude is a direct consequence in the discontinuity in the response. Therefore any slight error in fitting the boundary would trigger these green error spikes.

Let us we come back to the original objectives in fitting the response within optimization problem context. We can see that this approximation approach is a success as it complies with all requirements formulated previously:

- The accuracy of predicted values: with the exception of the high ro/low Afan corner, the error in predictions is well within the error amplitude of the original model
- The absence of ripple caused by the regression: By braking down the design space into zones, discontinuities in the response behavior could be captured by fairly simple nets and regressions. The association of these simple nets and regressions, allowed capturing with great accuracy the discontinuity in the response without artifically causing ripples (due to overfiting).
- Ability to capture both continuous values and discrete: The design space breakdown allowed isolating dysfunctional designs. This approach attemps to capture the cause rather than the consequence. By capturing this phenomenon independently from effects allows capturing this discontinuity with great accuracy while using a simple regression model.
- Avoid the creation of discontinuity in first order derivatives with no physical meaning: the transition weighting approach protects the regression from any

discontinuity between zone 1 and 2. It is important to note that this transition approach has not perceptible impact on the accuracy of the regression in zone 2.

B.4.4 Overview of the sizing model

The range of application of this model is defined on the thrust requirement. The range of application of the physics based model is limited by:

- The radius of the motor can vary between 65mm and 320 mm. The 65 mm limit was arbitrary. On the other hand the upper limit of 320 mm is based on a physical limit of the HTS model. Beyond this limit, the model can not return real operational limits (PowMax, TorMax and Nmax).
- The fan range extended between 0.2 and 9 m^2. These limits were arbitrary and were used as a reasonable range to which the performance of the E^3 fan map could be extrapolatedused.
- The aspect ratio of the motor was bounded between 1.6 and 8. The lower bound is defined by a geometric constraint from the HTS model. The upper constraint was arbitrary.

Limited to these design variable ranges the engine designs produced by the model will only be capable to perform over a certain range of thrust requirements. As a result the sizing model, which recieves thrust as input, is limited in the requirements it can satisfy. The maximum thrust requirement that can be considered for sizing was determined using the fan and motor pre-sizing. If the pre-sizing models can both find a pre-sized engine design, it means that there is at least one solution (however poor it will be).

The neural net- based regresssion (on which the optimizer is operating) can be trained for any arbitrary flight conditions. For simplicity sake, the range of application described in this document will be limited to the flight conditions considered in the thesis. These flight conditions are listed in the following table.

Table 90: Typical flight conditions

	Altitude	Mach #
1	0	0.04
2	50	0.41
3	275	0.75
4	350	0.78
5	150	0.71
6	100	0.54
7	100	0.45
8	0	0.3

In order to determine the maximum thrust that can be pre-sized for each scenario listed above a routine was defined (`limitscanner.m`). The regressed fan pre-sizing routine identifies the minimum fan area which can meet the thrust requirement. This fan area is then fed into the net capturing the coordinate of the boundary 12 which returns the minimum unconstrained motor radius. The maximum thrust requirement that can be satisfied by the sizing model is found when either:

- the pre-sized fan area is greater than 9 m2
- or, the pre-sized motor radius exceeds 320 mm.

Note these values are defined for the default ARm (which is defined as 4 by default).

The limiting thrust requirements for the flight conditions considered in the thesis are listed in the following table.

Table 91: Thrust limits for sizing model

	Altitude	Mach #	LimTh
	FL	[-]	[kN]
1	0	0.04	174
2	50	0.41	110
3	275	0.75	44
4	350	0.78	32
5	150	0.71	73
6	100	0.54	90
7	100	0.45	92
8	0	0.3	139

Observation of results

In order to observe the effect of the amplitude of the thrust constraints, the requirements associated with the basic mission considered in the thesis were scaled and submitted iteratively to the model.

$$OEC(\overline{x}) = \gamma_1 \frac{m_{engine}(\overline{x})}{m_{engine-baseline}} + \gamma_2 \frac{diameter(\overline{x})}{diameter_{baseline}} + \gamma_3 \frac{1-\eta_{CZ}(\overline{x})}{1-\eta_{CZ-baseline}} + \gamma_4 \frac{PowEreq(\overline{x})_{max}}{PowEreq|_{max-baseline}}$$

Optimization priority	γ_1	γ_2	γ_3	γ_4
Balanced	0.25	0.25	0.25	0.25
Power Optimized	0.25	0	0.25	0.5
Energy optimized	0.25	0	0. 5	0.25
Aggressive power optimization	0	0	0.5	0.5
Engine weight optimization	1	0	0	0

Table 92: Notional optimization strategies

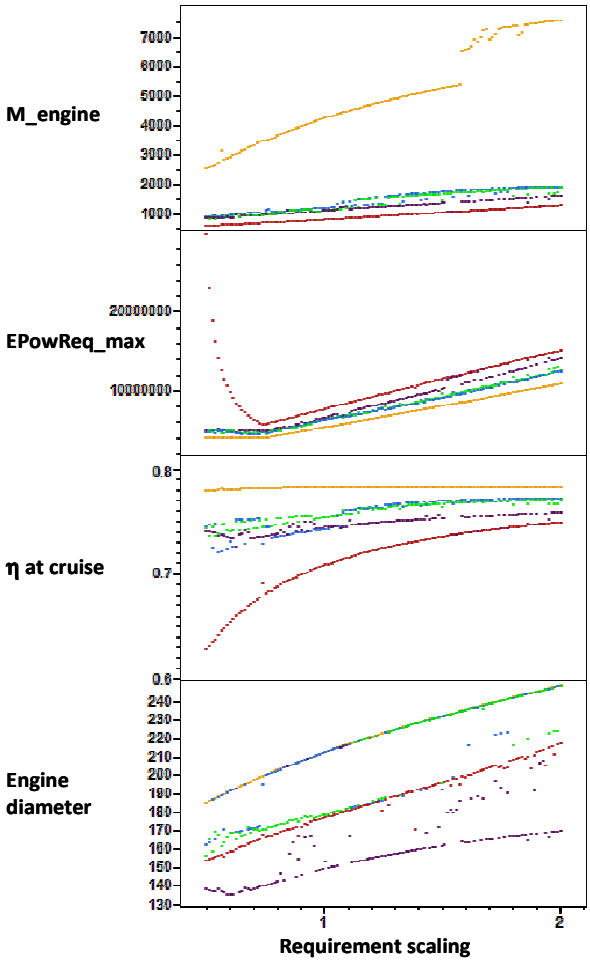


Figure 283: Sized designed scaled by requirements

Figure 146 presents this evolution for five different optimization strategies (each color represents points sized with a different strategy). The five strategies and their respective color code used to produce these lines are defined in Table 23. From these plots, we can observe the typical growth of the engine attributes as a function of the thrust requirements. But these plots also highlight the fact that the growth of the engine is strongly conditioned by the optimization strategy.

In order to observe how the sizing model adapts its solution, an exploration of the priority effects was performed. The objective function considered by the optimizer (presented above) is composed of 4 parameters. The priority factors associated with each of these four parameters were explored by performing a full factorial DoE. Each experiement in the DoE corresponded to a specific optimization problem with identical constraints but with different objective functions. The following scater plot represents the ensemble of sized design solutions which were returned by the model. Every point in this scater plot represents a feasible solution (i.e. a design which meets the thrust required by the mission). The line in green represents the Pareto front.

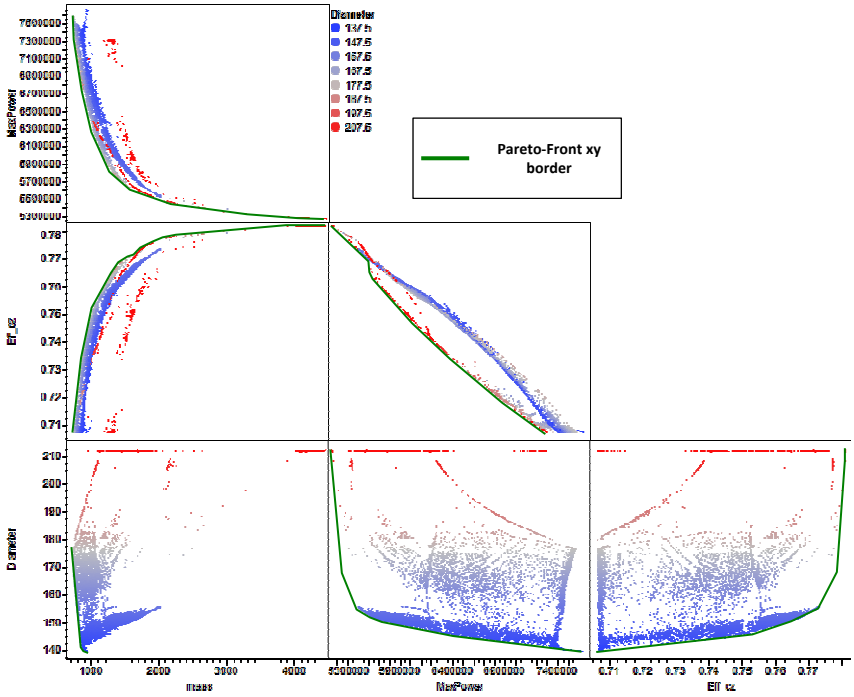


Figure 284: Overview of internal trade-offs

Figure 147 presents a number of design solutions to the same thrust requirements (i.e. aircraft mission). The great diversity in the attributes of the solutions presented in this figure highlights the need to perform an optimizer-based sizing process. The model proposed herein therefore does not only identify an engine design appropriate for the requirements. Granted that the appropriate objective function is provided, it can identify the "best" electric ducted fan design for the architecture.

B.5 Miscellaneous formulas and code

B.5.1 Isentropic relationships

Total properties

(119)

$$\frac{To}{T} = 1 + \frac{\gamma + 1}{2} M^2$$

(120)

$$\frac{Po}{P} = \left[1 + \frac{\gamma + 1}{2} M^2 \right]^{\frac{\gamma}{\gamma - 1}}$$

(121)

$$\frac{\rho o}{\rho} = \left[1 + \frac{\gamma + 1}{2} M^2 \right]^{\frac{1}{\gamma - 1}}$$

B.5.2 Flow function

This formulation of the flow function was provided by Mark Waters as part of his propulsion class at the Georgia institute of technology. The flow function (noted WFF) is a function of the Mach number only. At the same time, the value of the flow function is equal to a ratio of the area and the total properties (Po and To). This relationship is derived from the conservation of mass and assumes the air is a calorifically perfect gas.

(122)

$$WFF(M) = \frac{W \times \sqrt{To}}{A \times Po} = \sqrt{\frac{\gamma}{R}} \times M \times \left(1 + \frac{\gamma - 1}{2} M^2\right)^{-\frac{\gamma+1}{2(\gamma-1)}}$$

For R = 286.9 J/(kg.K) and $\gamma = 1.4$, a regression was build to fit the inverse of the flow function noted arcWFF.

$$M = arcWFF\left(\frac{W \times \sqrt{To}}{A \times Po}\right) = \left[\frac{\arcsin\left(\left(\frac{W \times \sqrt{To}}{A \times Po \times WFF(1)}\right)^{C1}\right)}{\pi / 2}\right]^{C2}$$

Where:

- WFF(1) = 0.040425 (kg K0.5)/(N.s) is the flow function value corresponding to M=1
- C1 = 0.614029378249447
- C2 = 1.50426194883171

A representation of the actual and regressed value is shown in the following figure:

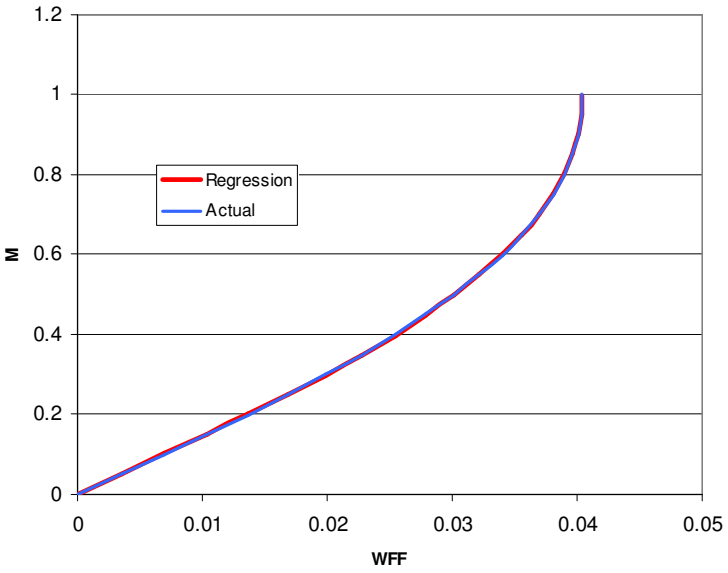


Figure 285: Illustration of the inverse flow function regression

B.5.3 Code

HSTlimits

```
function [OpLim,Dim]= HSTlimits(DVMotor)

% Model based on sizing model by Philippe Masson
% Translated and addapted as a performance model by Cyril de Tenorio

%% Inputs:
%     Ks = DVMotor(1);            % Armature Ampere turn loading (kA/m)
%     ARm = DVMotor(2);           % Machine shape factor (La/ro)
%     eis = DVMotor(3);           % Electrical radial airgap (mm) (5 mm is pretty small)
%     Ispe = DVMotor(4);          % Armature current density (Arms/mm2)
%     n2p = DVMotor(5);           % Number of pole pairs
%     ToP = DVMotor(6);           % Temperature (K)
%     ro = DVMotor(7);            % Mean armature radius  ro (mm)
%
%% Outputs:
% OpLim: Operational limits of the motor
%   1- Powmax[W]:   maximum power for the described motor
%   2- TorMax[Nm]:  maximum torque for the described motor
%   3- Nmax[RPM]:   maximum Speed for the described motor
% Dim: Dimention of the motor (assumes cylindrical shape)
%   1 - Ltot[m]:    Length of the motor
%   2 - re[m]:      Radius of the motor

Testingmode=0;      % should be 0 unless you are testing the function
%% for testing Make sure you comment the function line
if Testingmode~=0
    %% Input from the fan
    clc
    clear all
%     PowShaft = 2705 * 10 ^3;    % Power extracted on the shaft by fan [W]
%     NShaft = 9781;              % Speed of the shaft [rpm]

    %% Design variables
    Ks = 299.6;                 % Armature Ampere turn loading (kA/m)
    ARm = 1.63;                 % Machine shape factor (La/ro)
    eis = 45.4;                 % Electrical radial airgap (mm) (5 mm is pretty small)
    Ispe = 93.725879;           % Armature current density (Arms/mm2)
    n2p = 4;                    % Number of pole pairs
    ToP = 20.0;                 % Temperature (K)
    ro = 400;                   % Mean armature radius  ro (mm)
else
    Ks = DVMotor(1);            % Armature Ampere turn loading (kA/m)
    ARm = DVMotor(2);           % Machine shape factor (La/ro)
    eis = DVMotor(3);           % Electrical radial airgap (mm) (5 mm is pretty small)
    Ispe = DVMotor(4);          % Armature current density (Arms/mm2)
    n2p = DVMotor(5);           % Number of pole pairs
    ToP = DVMotor(6);           % Temperature (K)
    ro = DVMotor(7);            % Mean armature radius  ro (mm)
end
% Set by default
eisMin = 15;                % Minimum electrical airgap (mm)
te = 8;                     % Armature backiron radial standoff from armature (mm)
Jco = 150;                  % Critical current density (77K, 0T) Jco (A/mm2)
HSTopF = 0.6;               % HTS operating factor
HSTfr = 0.8;                % HTS filling ratio
beta = 120*pi/180;          % HTS angular aperture (rad)
PhiBI = 2.2;                % Magnetic flux density in the back iron (T)
%CryEff = 30;                % Carnot efficiency (cryocooler) [%]
%SpeWCryGVB=5/1643.9868;     % GVB cryocooler [kg/W]
HSTden = 8500;              % HTS density (kg/m3)
%Armden = 8500;              % Armature density (kg/m3)
%PhiL = 1.28;                    % Iron specific losses GVB (1.5 T, 50 Hz) 4 mil Hiperco
50A(W/kg)

%ArmOpFactor    = 0.6;          % Armature operating factor
%ArmLength4loss = 12;       % Armature AC loss dimension (µm,mils)

Steelden = 8321.369721;                 % density of steel     (kg/m3)
Steelstrain = 0.1;                      % strain                (%)
SteelModulus = 30000000*6894.75729;     % modulus               (psi)
Sigma_Op = Steelstrain/100*SteelModulus;% operating stress      (N/m2)
```

```
% Limits
BroLim = 1.3;
BrWindLim = 3.5;
r1tLim = 0.85;
reLim = 600;


%% Calculations
es = pi/3*Ks/Ispe;                          % Armature thickness es (mm, in)

%% Geometry
% HTS (rotor) outside radius (mm) r2
r2 = ro-eis-es/2;

% Back iron inner radius (mm) rs
rs = ro + es/2 + te;

% Active length (mm) La
La =ARm*ro;

% Total length (mm) Ltot
Ltot = La+pi*ro/n2p;

if ARm<2*pi/n2p || ro<eis-pi/6*Ks/Ispe %Constraints 1 and 2
    Tormax=0;
    Nmax=0;
    Powmax=0;
    re=0;
else

%% Definition of Coefficients
% Specific radial no-load field per torque [T/Nm] (Bro =  Cro * Torque)
Cro = 1000000/(1.4142*Ks *pi*((ro^3)*ARm));

% Ratio max field to no load field [-] (BrWind = Com * Bro)
Com = (ro/r2)^(n2p+1)*(1+(r2/rs)^(2*n2p))/(1+(ro/rs)^(2*n2p));

% Synchronous reactance from Bro (SynReac = Csr/Bro)
Csr = 0.0004*pi*Ks*(1+(ro/rs)^(2*n2p))/(1.4142);

% Back iron thickness coefficient (tiron = Cti * Bro * (1+SynReac^2)^.5)
Cti =2*rs*(ro/rs)^(n2p+1)/n2p/(1+(ro/rs)^(2*n2p))/PhiBI;

% Constant for field winding current density (Jf = Cjf*(Sum(a_i*T^i)))
Cjf = HSTopF*HSTfr*Jco*(1.4235-0.022467*ToP+0.000058307*ToP^2);

% Constant for r1t calculation (r1t =  (C1t * Cro * T)/(Cjf * Jf)
C1t= (n2p+2)*(ro/r2)^(n2p+1)/(1+(ro/rs)^(2*n2p))/(0.0008*r2*sin(beta/2));

Lim3 = BroLim/Cro;
Lim4 = BrWindLim/(Cro*Com);
Lim6 = ((reLim-rs)^2 - (Csr*Cti)^2)^.5 / (Cro*Cti);

TorMax = min([Lim3,Lim4,Lim6]);

%% Solving for the Torque constraint associated with r1temp
r1temp = HSTr1t(Cjf, C1t, Cro, Com, TorMax);
if r1temp<r1tLim
    Lim5 = TorMax;
else

    MaxIt = 100;
    Sinit = TorMax/10;
    Sacc = 0.1;

    TorMin = 0;
    TorBest = 0;
    it = 1;
    Tor = zeros(MaxIt,1);
    S = zeros(MaxIt,1);
    Test = zeros(MaxIt,1);
    S(1) = Sinit;   %Initial step
    Tor(1) = TorMax;  %Initial guess
    Table=zeros(MaxIt,3);
    %-02
    while (it<MaxIt)&&(S(it)>Sacc)&&(TorBest~=TorMax)
        %-03
        Table(it,3)=HSTr1t(Cjf, C1t, Cro, Com, Tor(it));
        Validity=(Table(it,3)<r1tLim);
```

```
        %-04
        if Validity
            %-06
            if Tor(it)>TorBest
                %-07
                TorBest = Tor(it);
            end
            %-08
            Test(it) = 1;
        else
            %-09
            Test(it) = -1;
        end
        %-10
        if it~=1 && Test(it)*Test(it-1)<0
            %-11
            S(it+1) = S(it)/(1.61803399*2);
        else
            S(it+1) = S(it);
        end
        %-13
        if Validity==1
            %-15
            Tor(it+1) = min(Tor(it) + S(it+1),TorMax);
        else
            %-14
            Tor(it+1) = max(Tor(it) - S(it+1),TorMin);
        end
        %-16
        it=it+1;
    end
    %-17
    Lim5 = TorBest;
    %Return is implicit
end
%% The maximum torque is based on the most constraining  limit
Tormax = min([Lim3,Lim4,Lim5,Lim6]);

%% Speed constraint resolution
% Based on the max Torque value the inner HTS winding radius is determined
r1temp = HSTr1t(Cjf, C1t, Cro, Com, Tormax);
% Rotor HTS inside radius (mm) r1
r1 = r2*(1-r1temp)^(1/(n2p+2));
% Based on r1 and the other dimensions of the motors it is possible to
% define the weight of the rotor
W_rotor = HSTden*HSTfr*beta*(r2*r2-r1*r1)*(La+pi/2*(pi-beta/2)*...
    (r2+r1)/2/n2p)*0.000000001;
% Note: The weight of the rotor defines the size of the containment tube
% which holds the rotor againts centripetal forces. This containment tube
% is geometrically constrained as it needs to fit in the airgap.

% Now that all geometrical attributes of the motor are defined we can
% derive the maximum speed
CmM = (Steelden*(r2/1000)^2*((1/60)*2*pi)^2)/Sigma_Op;
Ctcont = (1/Steelden)/(pi*2*r2/1000*Ltot/1000)*1000;
tcontmax = eis - eisMin - r2/10;

Nmax = (tcontmax/(CmM*(tcontmax+(Ctcont*W_rotor))))^.5;

Powmax = Tormax*5.1138e+003*60/(2*pi)*0.5; %%%%%%%%% THE COEFFICIENT IS ARBITRARY
% the calculation above is based on the max necesssary speed from the fan
% at M=0.5 alt=40,000ft. The arbitrary coefficient following it corresponds
% to the limit on power coming from the unknow phenomenon in the motor.

%% Solving for dimension of the motor
SynReac = Csr/(Cro * Tormax);
tiron = Cti * Cro * Tormax * (1+SynReac^2)^.5;
re = rs+tiron;
end

OpLim= [Powmax,Tormax,Nmax]; % Units [W,Nm,RPM]
Dim = [Ltot,re]/1000;        % Units [m,m]
```

FanModel

```
function                                                [Areas,M,P,Prop_e,Thrust,Choked,PowerProp,
PowerShaft]=FanModel(M_a,Alt,DVDuct,MotorCompDim,FanOpline,Ncs,Pplot)
%% subfunctions necessary to run this function
% ChokeArea
% ExhaustSolution
% FanMap
% FanTotTempRatio
% FullExpMach
% ISAPressure
% ISATemp
% MachAftFan
% MachAtArea
% Max_Mdot
% PlotEngine
% StagPressure
% StagTemp
% TotalPressure
% TotalTemp

%% Output
% Areas, M, P describe the duct and flow properties at each stage of the
% duct. Therefore their information is stored in a matrix where each column
% refers to a station (
% - Areas : [Aa  , A_in, A_fan, A_e]
% - M     : [M_a , M_1 , M_2  , M_3 , M_e]
% - P     : [Po_a, Po_1, Po_2 , Po_3, Po_e
%            P_a , P_1 , P_2  , P_3 , P_e]
%
% Prop_e  : exhaust properties
%           [v_e,M_e,T_e,To_e,P_e,Po_e,A_e]
%
% Thrust :  Thrust information
%           1: Net Thrust
%           2: Ram drag ~ m_dot * (- v_a)
%           3: Momentum thrust ~ m_dot * (v_e)
%           4: Pressure thrust ~ (P_e - P_a) * A_e
%
% Choked :  Signal for choked fan (impossible scenario of operation if ~=0)
%
% PowerProp:Propulsive power ~ Thrust(1)*v_a
%
% PowerShaft:Power necessary from the shaft to operate the fan

Testingmode=0;      % should be 0 unless you are testing the function
%% for testing Make sure you comment the function line
while Testingmode~=0
    clc
    clear all
    close all
    % Mission Input
    M_a = 0.5;
    Alt = 0;

    %Geometric Input
    A_fan = 1;
    rA_eMin = .0; %Ratio of A_fan
    rA_eMax = .75;%Ratio of A_fan
    rA_in = .941265;%Ratio of A_fan
            %1     %2     %3     %4
    DVDuct=[A_fan;rA_in;rA_eMin;rA_eMax];
    MotorCompDim(1) = 0.3; %[m] % Length
    MotorCompDim(2) = 0.25; %[m] % radius

    % Operation input
    FanOpline = 2;
    Ncs = .6;

    Pplot = 1;
    Testingmode=0;

end % Testing mode
%% Initialization
Areas = zeros(1,4);
Thrust = zeros(1,4); %[Total Thurst,Ram drag/Suction Thrust,Momentum thrust, Backpressure
Thrust]
M = zeros(1,5);
```

```
P = zeros(2,5);
Prop_e = zeros(1,7);
PowerProp = 0;
PowerShaft = 0;

%% Gas constant
Rair = 286.9; %[J/kg/K]
ya=1.4001;
Cpair = 1004; %[J/kg/K]

%% Design Variables
A_fan = DVDuct(1);
A_eMin = DVDuct(3)*A_fan;
A_eMax = DVDuct(4)*A_fan;
A_in = DVDuct(2)*A_fan;
Aref = FanMap(0,FanOpline,8);
FanScale = DVDuct(1)/Aref;
Wcs = FanMap(Ncs,FanOpline,4);

%% Mission
T_a = ISATemp(Alt);
P_a = ISAPressure(Alt);
r_a = P_a/(Rair*T_a);
a_a = (ya*Rair*T_a)^.5;
v_a = M_a*a_a;
To_a = TotalTemp(ya,T_a,M_a);
Po_a = TotalPressure(ya,P_a,M_a);
Del = Po_a/101325;
Thet = To_a/288.15;
m_dot = Wcs*Del/Thet^.5 * FanScale;


%% Inlet Choking conditions
m_dotMax=Max_Mdot(Rair,ya,Po_a,To_a,A_in);
Choked=0;
if m_dot>m_dotMax
    Choked = 1;
elseif Ncs<FanMap(1,2,5)
    Choked = -1;
elseif Ncs>FanMap(1,2,6)
    Choked = 4;
end
if Choked ~=0
    Thrust = [0,0,0,0];
else
    %% Inlet
    Aa = m_dot/(r_a*v_a);
    AxI = ChokeArea(ya,M_a,Aa);

    M_1 = MachAtArea(AxI,A_in,ya);
    %To_1 = To_a;
    %T_1 = StagTemp(ya,To_1,M_1);
    %a_1 = (ya*Rair*T_1)^.5;
    %v_1 = a_1*M_1;
    Po_1 = Po_a;
    %ro_1 = Po_1/(Rair*To_1) ;
    %r_1 = StagDens(ya,ro_1,M_1);
    P_1 = StagPressure(ya,Po_1,M_1);

    M_2 = MachAtArea(AxI,A_fan,ya);
    if M_2==1
        Thrust = [0,0,0,0];
        Choked = 2;
    else
        To_2 = To_a;
        %T_2 = StagTemp(ya,To_2,M_2);
        %a_2 = (ya*Rair*T_2)^.5;
        %v_2 = a_2*M_2;
        Po_2 = Po_a;
        %ro_2 = Po_2/(Rair*To_2) ;
        %r_2 = StagDens(ya,ro_2,M_2);
        P_2 = StagPressure(ya,Po_2,M_2);

        %% Fan
        Pie23 = FanMap(Ncs,FanOpline,2);
        Eff23 = FanMap(Ncs,FanOpline,3);
        To_3 = To_2*FanTotTempRatio(Eff23,Pie23,ya);
        Po_3 = Po_2*Pie23;
        %ro_3 = Po_3/(Rair*To_3);
        ActualFlux = m_dot/A_fan;
```

```
        M_3 = MachAftFan(ActualFlux,Po_3,To_3,Rair,ya);
        %T_3 = StagTemp(ya,To_3,M_3);
        %r_3 = StagDens(ya,ro_3,M_3);
        P_3 = StagPressure(ya,Po_3,M_3);
        %a_3 = (ya*Rair*T_3)^.5;
        %v_3 = M_3*a_3;

        %% Nozzle
        To_e = To_3;
        Po_e = Po_3;
        %ro_e = ro_3;
        AxN = ChokeArea(ya,M_3,A_fan);
        Mexp = FullExpMach(ya, Po_e,P_a);
        [A_e,M_e]=ExhaustSolution(ya,Mexp,AxN,A_eMin, A_eMax);
        P_e = StagPressure(ya,Po_e,M_e);
        T_e = StagTemp(ya,To_e,M_e);
        %r_e = StagDens(ya,ro_e,M_e);
        a_e = (ya*Rair*T_e)^.5;
        v_e = M_e*a_e;

        if M_e>=1 && A_e >= A_eMax
            Thrust = [0,0,0,0];
            Choked = 3;
        else
            Choked = 0;
            Thrust(2) = m_dot * (- v_a);
            Thrust(3) = m_dot * (v_e);
            Thrust(4) = (P_e - P_a) * A_e;
            Thrust(1) = Thrust(2)+Thrust(3)+Thrust(4);
            PowerProp = Thrust(1)*v_a;
            PowerShaft = FanPower(Cpair, Eff23, ya, Pie23, To_2, m_dot);
        end
        NcsMax_Best=Ncs;
        %1   2   3   4    5    6   7
        Prop_e=[v_e,M_e,T_e,To_e,P_e,Po_e,A_e];
        M = [M_a,M_1,M_2,M_3,M_e];
        P = [Po_a,Po_1, Po_2, Po_3,Po_e
            P_a,P_1 P_2,P_3,P_e];
        Areas = [Aa, A_in,A_fan,A_e];
        %% plotting results
        if Pplot~=0

PlotEngine(1,Pplot,DVDuct,MotorCompDim,FanOpline,Alt,Areas,M,P,Thrust,Ncs,PowerProp,Power
Shaft,0,ya);
        end
    end
 end
```

TestConstraints

```
function
Validity=TestConstraints(Ncst,DVDuct,MotorCompDim,Del,Thet,ya,Powlim,Torlim,Opline,Plotto
ken)
%%Output
% Validity: if this variable is 0 then the operating point is infeasible
%                               1 then the operating point is feasible

Testingmode=0;      % should be 0 unless you are testing the function
%% for testing Make sure you comment the function line
if Testingmode~=0
    clc
    clear all
    close all
    Ncst =0.9130; % Speed to be tested

    % Duct, Motor & Fan Design Variables
    A_fan = 0.6573;
    A_eMin = .2; %Ratio of A_fan
    A_eMax = .7;%Ratio of A_fan
    A_in = .98;%Ratio of A_fan
             %1     %2     %3     %4
    DVDuct=[A_fan;A_in;A_eMin;A_eMax];
    MotorCompDim(1) = 0.3; %[m] % Length
    MotorCompDim(2) = 0.25; %[m] % radius
```

```
    % Mission Input
    M_a = .01;
    Alt = 0;
    Powlim = 6.1427e+006; % [W]
    Torlim =  1.7235e+004; % [Nm]
    %Nlim = 7000 * 60/(2*pi); % [RPM]
    Opline = 1;
    Plottoken = 2;

    ya=1.4;
    T_a = ISATemp(Alt);
    P_a = ISAPressure(Alt);
    To_a = TotalTemp(ya,T_a,M_a);
    Po_a = TotalPressure(ya,P_a,M_a);
    Del = Po_a/101325;
    Thet = To_a/288.15;
end

% Gas constants
Cpair = 1004;
% gamma = 1.4;
Tref = 288.15;

%% Initialization
Afan = DVDuct(1);                       % Fan area
Aemax= DVDuct(4)*Afan;                  % Max exhaust throat area
Ai =  DVDuct(2)*Afan;                   % Inlet throat area
Aref = FanMap(0,Opline,8);              % Reference map fan area
rref = FanMap(0,Opline,9);              % Reference map fan radius
rtip = (MotorCompDim(2)^2+Afan/pi)^.5;  % Fan tip radius
Nnom = FanMap(0,2,7);                   % Fan nominal speed
%% Operation point to be tested
Wcst=FanMap(Ncst,Opline,4);
Prt=FanMap(Ncst,Opline,2);
Efft=FanMap(Ncst,Opline,3);
%% inlet constraint
WcsMaxInlet = 241.303 *Aref*Ai/Afan;
if WcsMaxInlet<Wcst
    Validity = 0;
else
    % Choked exhaust constraint
    PrCst = Prt/(1+(Prt^((ya-1)/ya)-1)/ Efft)^.5;
    % Term as function of pie from equation 28 in report
    WcsMaxExhaust = PrCst*241.303 *Aref*Aemax/Afan;
    if WcsMaxExhaust<Wcst
        Validity =0;
    else

PrTorlim=(1+Efft*Torlim*Ncst/(Cpair*Wcst*Del*Tref)*(Aref/Afan*rtip/rref)*(2*pi*Nnom/60))^
(3.5);

PrTorlim=(1+Efft*Torlim*Ncst/(Cpair*Wcst*Del*Tref)*(Aref/Afan*rtip/rref)*Nnom/(Thet^.5))^
(3.5);
        PrPowlim=(Efft*Powlim/(Cpair*Wcst*Tref*Thet^.5*Del)*(Aref/Afan)+1)^(3.5);

        if min(PrPowlim,PrTorlim)<Prt
            Validity = 0;
        else
            Validity =1;
        end
    end
end

%% Building Plot
if Plottoken~=0
    % Loading map values
    Wcs = FanMap(0,Opline,12); % Loading map data point for the mass flow
    Prf = FanMap(0,Opline,13); % Loading map data point for the pressure ratio

    % Speed lines and overspeed constraint
    for sp=1:10
        hold on
        if sp<3
            plot(Wcs(1:6,sp),Prf(1:6,sp),'-k')
        elseif sp<10
            plot(Wcs(:,sp),Prf(:,sp),'-k')
        else
            plot(Wcs(:,sp),Prf(:,sp),'-r')
        end
    end
```

```
    % Surge and windmill constraints
    plot(Wcs(1,1:10),Prf(1,1:10),'-r')
    plot(Wcs(7,3:10),Prf(7,3:10),'-b')

    % Inlet constraint
    plot([WcsMaxInlet,WcsMaxInlet],[1,2],'-m')

    % Opline and exhaust constraint
    Nopl =linspace(.4,1.05,100);
    Wline=zeros(1,100);
    Prline = zeros(1,100);
    Effline = zeros(1,100);
    WcsMaxExhaust= zeros(1,100);
    for sp=1:100
        Wline(sp)=FanMap(Nopl(sp),2,4);
        Prline(sp)=FanMap(Nopl(sp),2,2);
        Effline(sp)=FanMap(Nopl(sp),2,3);
        PieContr = Prline(sp)/(1+(Prline(sp)^((ya-1)/ya)-1)/ Effline(sp))^.5;
        % from equation 28 in report
        WcsMaxExhaust(sp) = PieContr*241.303 *Aref*Aemax/Afan;
    end
    plot(WcsMaxExhaust,Prline,'-m')
    plot(Wline,Prline,'-g')
    axis([200 800 1 2])

    % Power constraint
    PrPowlim = zeros(1,100);
    PrTorlim = zeros(1,100);

    for sp=1:100

PrPowlim(sp)=(Powlim/(Thet^.5*Del)*Aref/Afan*Effline(sp)/(Cpair*Wline(sp)*Tref)+1)^(3.5);

PrTorlim(sp)=(1+Effline(sp)*Torlim*Nopl(sp)/(Cpair*Wline(sp)*Del*Tref)*(Aref/Afan*rtip/rr
ef)*(2*pi*Nnom/60))^(3.5);

%PrTorlim(sp)=(Torlim*Nopl(sp)*Nnom/(Thet^.5*Del)*(Aref/Afan*rtip/rref)*Effline(sp)/(Cpai
r*Wline(sp)*Tref)+1)^(3.5);
    end
    plot(Wline,PrPowlim,'-c')
    plot(Wline,PrTorlim,'-b')
    hold off
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function r1t = HSTr1t(Cjf, C1t, Cro, Com, T)

BrWindM = Com * Cro * T;
Jf=Cjf*(5.22-1.5166*BrWindM+0.353*BrWindM^2-0.037625*BrWindM^3+0.0014773*BrWindM^4);
r1t = C1t*Cro*T/Jf;
```

MaxTrustCap3

```
function
[Thrust,NcsBest,Properties,PoStages,ShaftOp]=MaxTrustCap3(DVDuct,MotorCompDim,MotorLim,M_
a,Alt,Pplot)
%%Ouputs
%Thrust = net thrust produced by engine [N]
%NcsBest =  Max speed achievable by the engine [-]
%Properties=[1x8][v_e,M_e,T_e,To_e,P_e,Po_e,A_e,Prop_eff];
%PoStages = [1x5][Po_a, Po_i, Po_2, Po_3, Po_e] [Pa]
%ShaftOp= [3x1]
%   - 1: Normalized speed at max thrust [N]
%   - 2: Actual speed at max thrust [RPM]
%   - 1: Normalized speed at max thrust


Testingmode=0;      % should be 0 unless you are testing the function
%% for testing Make sure you comment the function line
if Testingmode~=0
```

```
    clc
    clear all
    close all
    % Mission Input
    M_a = 0.01;
    Alt = 0;

    %Geometric Input
    A_fan = 0.6573;
    rA_eMin = .2; %Ratio of A_fan
    rA_eMax = .6;%Ratio of A_fan
    rA_in = .941265;%Ratio of A_fan
             %1     %2     %3     %4
    DVDuct=[A_fan;rA_in;rA_eMin;rA_eMax];

    MotorCompDim(1) = 0.3; %[m] % Length
    MotorCompDim(2) = 0.25; %[m] % radius

    PowLim = 18e6;
    Tormotor = 8200;
    Nlim = 800 * 60/(2*pi);
    MotorLim=[PowLim,Tormotor,Nlim];

    % Operation input
    Pplot = 1;
end % Testing mode

%% Air constants%%
Rair = 286.9;
ya=1.4;

%% Fan reference values
FanOpline = 2;
rref = FanMap(0,FanOpline,9);           % Reference map fan radius
Nnom = FanMap(0,FanOpline,7);           % Reference map fan nom speed
% Aref = FanMap(0,FanOpline,8);         % Reference map fan area
Afan = DVDuct(1);
rtip = (MotorCompDim(2)^2+Afan/pi)^.5;  % Fan tip radius

%% Mission
T_a = ISATemp(Alt);
P_a = ISAPressure(Alt);
% r_a = P_a/(Rair*T_a);
a_a = (ya*Rair*T_a)^.5;
v_a = M_a*a_a;
To_a = TotalTemp(ya,T_a,M_a);
Po_a = TotalPressure(ya,P_a,M_a);

%% Correction and scaling parameters
Del = Po_a/101325;
Thet = To_a/288.15;

%% Fan/Duct/Motor Matching
MaxIt = 100;
Sinit = 0.05;
Sacc = 0.001;

%-01
% Max motor speed corrected to ambient conditions, scalled to reference fan
% and normalized by nominal speed
Ncsnlim = MotorLim(3)/(Thet)^.5 * rtip/rref/Nnom;
NcsMin = FanMap(0,FanOpline,5);
NcsMax = min(FanMap(0,FanOpline,6),Ncsnlim);          % The max speed is the smallest
between the fan and motor limiting speeds
NcsBest = 0;
it = 1;
Ncs = zeros(MaxIt,1);
S = zeros(MaxIt,1);
Test = zeros(MaxIt,1);
S(1) = Sinit;   %Initial step
Ncs(1) = NcsMax;  %Initial guess
%-02
while (it<MaxIt)&&(S(it)>Sacc)&&(NcsBest~=NcsMax)&&NcsMax>NcsMin
    %-03

Validity=TestConstraints(Ncs(it),DVDuct,MotorCompDim,Del,Thet,ya,MotorLim(1),MotorLim(2),
FanOpline,0);
    %-04
    if Validity==1
        %-06
```

```
        if Ncs(it)>NcsBest
            %-07
            NcsBest = Ncs(it);
        end
        %-08
        Test(it) = 1;
    else
        %-09
        Test(it) = -1;
    end
    %-10
    if it~=1 && Test(it)*Test(it-1)<0
        %-11
        S(it+1) = S(it)/(1.61803399*2);
    else
        S(it+1) = S(it);
    end
    %-13
    if Validity==1
        %-15
        Ncs(it+1) = min(Ncs(it) + S(it+1),NcsMax);
    else
        %-14
        Ncs(it+1) = max(Ncs(it) - S(it+1),NcsMin);
    end
    %-16
    it=it+1;
end
%-17
NcsMax_Best = NcsBest;
%Return is implicit

%% Run engine at max regime
[Areas,M,P,Prop_e,Thrustelements,Choked,PowerProp, PowerShaft]=...
    FanModel(M_a,Alt,DVDuct,MotorCompDim,FanOpline,NcsMax_Best,0);
%-17
Thrust=Thrustelements(1);
Properties = zeros(1,10);
Properties(1,1)=P_a;
Properties(1,2)=v_a;
Properties(1,3:9) = Prop_e;

%Stage total pressures
PoStages = P(1,:);

%Shaft properties
if PowerShaft==0
    Properties(1,10)= 0;
else
    Properties(1,10)= PowerProp/PowerShaft;
end
NShaft = NcsMax_Best*Thet^.5*Nnom*rtip/rref;
ShaftOp = [NcsMax_Best;NShaft;PowerShaft];


if Pplot~=0

PlotEngine(2,Pplot,DVDuct,MotorCompDim,FanOpline,Alt,Areas,M,P,Thrustelements,NcsMax_Best
,PowerProp,PowerShaft,MotorLim,ya);
end
```

Thrustpoint

```
function
[ShaftOp,PoStages,PartialThrustRatio,efficiency]=Thrustpoint(DVDuct,FanOpline,MotorCompDi
m,ThrustTarget,ShaftOpFullTh,M_a,Alt,Pplot)


Testingmode=0;      % should be 0 unless you are testing the function
%% for testing Make sure you comment the function line
if Testingmode~=0
    clc
    clear all
    close all

    %% Mission
    ThrustTarget = 36.2e3*0.3 ;
    M_a  = 0.6;
```

```
    Alt = 260*100;
    FanOpline = 2;
    %% Duct, Fan Design Variables
    A_fan =  0.6573;            % Fan Area
    rA_eMin = .0;               % Minimum exhaust area as a ratio of A_fan
    rA_eMax = .8;               % Maximum exhaust area as a ratio of A_fan
    rA_in = .95;                % Inlet throat area as a ratio of A_fan
    %1     %2     %3     %4
    DVDuct=[A_fan;rA_in;rA_eMin;rA_eMax];

    MotorCompDim(1) = 0.3; %[m] % Length
    MotorCompDim(2) = 0.25; %[m] % radius

    % Shaft mech properties at max power (Nnormalized, N shaft [RPM], Power shaft [W])
    ShaftOpFullTh=[1.05;3429.62651912916;9604897.99568038];
end%testing mode

%% Air constants%%
Rair = 286.9;
ya=1.4;

%% Mission
T_a = ISATemp(Alt);
P_a = ISAPressure(Alt);
% r_a = P_a/(Rair*T_a);
a_a = (ya*Rair*T_a)^.5;
v_a = M_a*a_a;
To_a = TotalTemp(ya,T_a,M_a);
Po_a = TotalPressure(ya,P_a,M_a);

% Correction and scaling parameters
Del = Po_a/101325;
Thet = To_a/288.15;
Aref = FanMap(0,FanOpline,8);              % Reference map fan area
rref = FanMap(0,FanOpline,9);              % Reference map fan radius
rtip = (MotorCompDim(2)^2+DVDuct(1)/pi)^.5;% Fan tip radius
Nnom = FanMap(0,2,7);                      % Fan nominal speed

%% Fan/Duct/Motor Matching
MaxIt = 100;
Sacc = 0.001;

%-01

NcsMin = FanMap(0,FanOpline,5); %Speed under which the fan become inefective
NcsMax = ShaftOpFullTh(1); %Speed at full throttle
NcsBest = 0;
it = 1;
Ncs = zeros(MaxIt,1);
S = zeros(MaxIt,1);
Test = zeros(MaxIt,1);
S(1) = (NcsMax - NcsMin)/5;   %Initial step
Ncs(1) = NcsMin;  %Initial guess

[Areas,M,P,Prop_e,Thrustelements,Choked,PowerProp, PowerShaft]=...
    FanModel(M_a,Alt,DVDuct,MotorCompDim,FanOpline,NcsMin,0);


if Thrustelements(1)>ThrustTarget % This condition checks that the minimum
    % active thrust from the map does not exceeds the target thrust
    NcsBest = NcsMin;
    efficiency = PowerProp/PowerShaft;
    PartialThrustRatio = ThrustTarget/Thrustelements(1);% if it does the
    % energy is calculated based on the assumption that the efficiency is
    % constant (i.e: we are shutting down some engines so that the iddling
    % engines provide the necessary thrust)
else
    %-02
    while (it<MaxIt)&&(S(it)>Sacc)&&(NcsMax>NcsMin)
        %-03
        [Areas,M,P,Prop_e,Thrustelements]=...
            FanModel(M_a,Alt,DVDuct,MotorCompDim,FanOpline,Ncs(it),0);
        Thrust=Thrustelements(1);
        if Thrust<ThrustTarget
            N2small=1;
            if Ncs(it)==NcsMax
                it=MaxIt;
                NcsBest=0;
            end
        else
```

```
            N2small=0;
        end
        %-04
        if N2small==1
            %-08
            Test(it) = 1;
        else
            %-06
            if NcsBest==0 || Ncs(it)<NcsBest
                %-07
                NcsBest = Ncs(it);
            end
            %-09
            Test(it) = -1;
        end
        %-10
        if it~=1 && Test(it)*Test(it-1)<0
            %-11
            S(it+1) = S(it)/(1.61803399*2);
        else
            S(it+1) = S(it);
        end
        %-13
        if N2small==1
            %-15
            Ncs(it+1) = min(Ncs(it) + S(it+1),NcsMax);
        else
            %-14
            Ncs(it+1) = max(Ncs(it) - S(it+1),NcsMin);
        end
        %-16
        it=it+1;
    end
    %-17
    % NcsBest = NcsBest;
    %Return is implicit

    % Operational properties at target thrust
    [Areas,M,P,Prop_e,Thrustelements,Choked,PowerProp, PowerShaft]=...
    FanModel(M_a,Alt,DVDuct,MotorCompDim,FanOpline,NcsBest,0);
    efficiency = PowerProp/PowerShaft;

    PartialThrustRatio = 1;
end

%Stage total pressures
PoStages = P(1,:);

% Shaft operation
NShaft = NcsBest*Thet^.5*Nnom*rtip/rref;
ShaftOp = [NcsBest;NShaft;PowerShaft];
```

HTSatt

```
function [PowEreq,W_totalPhilippe]= HTSatt(OpLim,ShaftOp,DVMotor)

Testingmode=0;      % should be 0 unless you are testing the function
%% for testing Make sure you comment the function line
if Testingmode~=0
    clc
    clear all
    close all
    %% Motor Design Variables
    Ks = 299.6;                 % Armature Ampere turn loading (kA/m)
    ARm = 3;                  % Machine shape factor (La/ro)
    eis = 45.4;                 % Electrical radial airgap (mm) (5 mm is pretty small)
    Ispe = 93.725879;           % Armature current density (Arms/mm2)
    n2p = 4;                    % Number of pole pairs
    ToP = 20.0;                 % Temperature (K)
    ro = 140;                   % Mean armature radius  ro (mm)

    DVMotor(1) = Ks;            % Armature Ampere turn loading (kA/m)
    DVMotor(2) = ARm;           % Machine shape factor (La/ro)
    DVMotor(3) = eis;           % Electrical radial airgap (mm) (5 mm is pretty small)
    DVMotor(4) = Ispe;          % Armature current density (Arms/mm2)
    DVMotor(5) = n2p;           % Number of pole pairs
    DVMotor(6) = ToP;           % Temperature (K)
    DVMotor(7) = ro;            % Mean armature radius  ro (mm)
```

```
    %% Motor attributes previously calculated
    % OpLim= Limiting operating conditions [Powmax,Tormax,Nmax]
    % Units [W,                    Nm,                    RPM]
    OpLim = [4407563424.81821,180515.062193020,12034.4226055166];

    %% Shaft operating conditions
    %ShaftOp(1,:,:)= Operation at full throttle (N normalized, N actual in RPM, Shaft
Power in [W])
    %ShaftOp(2,:,:)= Operation at nominal mission thrust (N normalized, N actual in RPM,
Shaft Power in [W])
    ShaftOp(1,:,:)                                                                    =
[0.921352549147123,0.900000000000000,1.05000000000000,1.05000000000000,1.05000000000000,0
.900000000000000,0.921352549147123

3434.66862611111,3319.83901375729,4171.75554181081,4198.73512645606,4171.75554181081,3319
.83901375729,3434.66862611111

9656567.07959170,9447321.25013789,6062331.41813680,5017733.27173750,6062331.41813680,9447
321.25013789,9656567.07959170];
    ShaftOp(2,:,:)                                                                    =
[0.817082039317698,0.800000000000000,0.920000000000000,0.920000000000000,0.92000000000000
0,0.800000000000000,0.817082039317698

3045.96329385664,2950.96801222870,3655.25247472947,3678.89172984722,3655.25247472947,2950
.96801222870,3045.96329385664

6325189.66935168,6286829.09416414,4082670.42047687,3379186.94861810,4082670.42047687,6286
829.09416414,6325189.66935168];
else
    Ks = DVMotor(1);            % Armature Ampere turn loading (kA/m)
    ARm = DVMotor(2);           % Machine shape factor (La/ro)
    eis = DVMotor(3);           % Electrical radial airgap (mm) (5 mm is pretty small)
    Ispe = DVMotor(4);          % Armature current density (Arms/mm2)
    n2p = DVMotor(5);           % Number of pole pairs
    ToP = DVMotor(6);           % Temperature (K)
    ro = DVMotor(7);            % Mean armature radius  ro (mm)
end % Testing mode

%% Technology factors (Set by default)
eisMin = 15;                % Minimum electrical airgap (mm)
te = 8;                     % Armature backiron radial standoff from armature (mm)
Jco = 150;                  % Critical current density (77K, 0T) Jco (A/mm2)
HSTopF = 0.6;               % HTS operating factor
HSTfr = 0.8;                % HTS filling ratio
beta = 120*pi/180;          % HTS angular aperture (rad)
PhiBI = 2.2;                % Magnetic flux density in the back iron (T)
CryEff = 30;                % Carnot efficiency (cryocooler) [%]
% SpeWCryGVB=5/1643.9868;     % GVB cryocooler [kg/W]
HSTden = 8500;              % HTS density (kg/m3)
Armden = 8500;              % Armature density (kg/m3)
PhiL = 1.28;                    % Iron specific losses GVB (1.5 T, 50 Hz) 4 mil Hiperco
50A(W/kg)

ArmOpFactor = 0.6;          % Armature operating factor
ArmLength4loss  = 12;       % Armature AC loss dimension (µm,mils)

Steelden = 8321.369721;                 % density of steel     (kg/m3)
Steelstrain = 0.1;                      % strain                (%)
SteelModulus = 30000000*6894.75729;     % modulus               (psi)
Sigma_Op = Steelstrain/100*SteelModulus;% operating stress      (N/m2)



%% Organizing operational information
Nmax = 0;
A=size(ShaftOp);
K=A(3);

for k=1:K
    for a=1:2
        Nmax = max(Nmax,ShaftOp(a,2,k));
    end
end

TorMax = OpLim(2);

%% Geometry
% Armature thickness es (mm, in)
es = pi/3*Ks/Ispe;
```

```
% HTS (rotor) outside radius (mm) r2
r2 = ro-eis-es/2;

% Back iron inner radius (mm) rs
rs = ro + es/2 + te;

% Active length (mm) La
La =ARm*ro;

% Total length (mm) Ltot
Ltot = La+pi*ro/n2p;

% Radial no load field at armature Bro (T)
Bro = 1000000*TorMax/(1.4142*Ks *pi*((ro^3)*ARm));

% Radial max field on HTS winding (T)
BrWind = Bro*(ro/r2)^(n2p+1)*(1+(r2/rs)^(2*n2p))/(1+(ro/rs)^(2*n2p));

% Critical current density (based on max radial field) (A/mm2)
JcMax = Jco*(5.22-1.5166*BrWind+0.353*BrWind^2-...
    0.037625*BrWind^3+0.0014773*BrWind^4)*(1.4235-0.022467*ToP+...
    0.000058307*ToP^2);
% Field winding current density (A/mm2)
Jf = HSTopF*HSTfr*JcMax;

% Intermediate calculation for r1
r1temp = Bro*(n2p+2)*(ro/r2)^(n2p+1)/(0.0008*Jf*r2*sin(beta/2))/...
    (1+(ro/rs)^(2*n2p));

% Rotor HTS inside radius (mm) r1
r1 = r2*(1-r1temp)^(1/(n2p+2));


% Synchronous reactance (p.u.)
SynReac = 0.0004*pi*Ks*(1+(ro/rs)^(2*n2p))/(1.4142*Bro);

% Back iron thickness (mm)
tiron =2*rs*Bro*(ro/rs)^(n2p+1)/n2p/(1+(ro/rs)^(2*n2p))/PhiBI*...
    (1+SynReac*SynReac)^.5;

% Back iron external radius re  (mm)
re = rs+tiron;

% Back iron weight (kg)
W_iron= 0.00000765*pi*(re*re-rs*rs)*La;

% Volume of superconductor (m3) !Added by Cyril
Vsc = 0.000000002*pi*ro*es*(La+5*pi*pi/12/n2p*ro);


%% Torque tube
% incr. ms/MrotorHTS    ( - )
IncrMshaft =(Steelden*(r2/1000)^2*((Nmax/60)*2*pi)^2)/Sigma_Op;

% Total str ms/MrotorHTS( - )
Sigma_tot = IncrMshaft/(1-IncrMshaft);

% Torque tube mass      (kg)
SingleTorTubeW = (((Ltot+2*r2)/1000)*TorMax*Steelden)/(Sigma_Op*r2/1000);

%% Weight Calculations
% Weight of torque tubes (2) (kg)
W_TorTube = 2 * SingleTorTubeW;

% Armature winding weight (kg)
W_wind = Armden*Vsc;

% Rotor HTS weight (kg)
W_rotor= HSTden*HSTfr*beta*(r2*r2-r1*r1)*(La+pi/2*(pi-beta/2)*...
    (r2+r1)/2/n2p)*0.000000001;

% Weight HTS total (kg)
W_HST = W_wind + W_rotor;

% Weight of rotor containment (kg)
W_rotorCont = W_rotor *Sigma_tot;

% Weight Motor/Gen Total (kg)
W_Motor = W_HST + W_iron + W_rotorCont + W_TorTube;
```

```
%% Losses and PowE requirements (scenario specific)
PowEreq = zeros(2,K);
HeatMax = 0;
for a=1:2
    for k=1:K
        % Speed in scenario k (RPM)
        Nk= ShaftOp(a,2,k);

        % Torque in scenario k (Nm)
        Tork = 30000*ShaftOp(a,3,k)/1000/pi/Nk;

        % Electrical frequency (Hz)
        freqE =Nk*n2p/60;

        % Radial no load field at armature Bro in scenarios k(T)
        Brok = 1000000*Tork/(1.4142*Ks *pi*((ro^3)*ARm));

        % Iron specific losses (BFe, f) (W/kg)
        BFe= PhiL/2*(freqE/50 + (freqE/50)^2)*(PhiBI/1.5)^(2);

        % Iron losses (W)
        Loss_iron = W_iron*BFe;

        % Armature Required? critical current density (A/mm2)
        Jca =(Ispe/ArmOpFactor)*1.414;

        % Volume of superconductor (m3) !Added by Cyril
        Vsc = 0.000000002*pi*ro*es*(La+5*pi*pi/12/n2p*ro);

        % Arm. AC losses (W) (at cold T)
        Loss_Arm= 8/3/pi*Jca*ArmLength4loss*freqE*Brok*Vsc;

        % Cooler Input Power for Armature AC losses (W)
        Loss_cool= Loss_Arm *(300-ToP)/ToP/(CryEff /100);

        % Thermal losses (W)
        Loss_th = 2;

        % AC field winding losses (W)
        Loss_field = 0;

        % Total field winding cryogenic losses (W)
        Loss_wind = Loss_th+Loss_field;

        % Heat to be extracted by cryocooler (W)
        Heat = Loss_Arm+Loss_wind;

        % Total electrical losses (W)
        Loss_TotElectic = Loss_cool+Loss_iron;

        PowEreq(a,k) = ShaftOp(a,3,k) + Loss_TotElectic;

        HeatMax = max(HeatMax,Heat);
    end
end

% Cryocooler mass (kg)
W_cryo = 157*(2.7183)^(-0.0533*ToP)*(HeatMax)^(0.009*ToP+0.1275);

% Weight Motor + Cryoc (kg)
W_MotorCry = W_Motor + W_cryo;

% Weight as calculated by Philippe model (kg)
W_totalPhilippe = W_MotorCry +(W_HST+W_iron)*1.6 - (W_Motor);
```

MassModel

```
function [m_engine,Dim_engine]=MassModel(DVDuct,MotorCompDim,PoMaxStages,m_motor,Pplot)

Testingmode=0;      % should be 0 unless you are testing the function
%% for testing Make sure you comment the function line
if Testingmode~=0
    clc
    clear all
    close all

    m_motor = 2.8487e+003;              % Motor + cooler mass           [kg]
```

```
    %% Duct, Fan Design Variables
    A_fan = 1;                  % Fan Area
    rA_eMin = .0;               % Minimum exhaust area as a ratio of A_fan
    rA_eMax = .8;               % Maximum exhaust area as a ratio of A_fan
    rA_in = .95;                % Inlet throat area as a ratio of A_fan
    %           1     %2     %3     %4
    DVDuct=[A_fan;rA_in;rA_eMin;rA_eMax];

    % Length & radius of the motor compartment in [m]
    MotorCompDim = [0.53,0.16];

    % Maximum pressure seen on the nacelle and compartment walls [Pa]
    PoMaxStages = [107136.047653181,107136.047653181,167286.645453275,167286.645453275];
    Pplot=1;

end

[xpos,rduct]=FanGeometry(DVDuct,MotorCompDim,0);

%% Estimation of the mass of the fan mechanical system (incl.bearings) [kg]
[Mass_Fan,MassMechElements] = FanMechMass(rduct);

%% Nacelle weights
[Mass_Nacelle,NacelleMassDecomposed] = MassWallDuct(xpos,rduct,PoMaxStages);

m_engine = m_motor+Mass_Nacelle+Mass_Fan;

if Pplot == 1
    Mass_fan = MassMechElements(1);
    Mass_fanframe = MassMechElements(3);
    Mass_shaft = MassMechElements(2);
    Mass_inlet = sum(NacelleMassDecomposed(1,1));
    Mass_fanCowl=sum(NacelleMassDecomposed(2,1));
    Mass_spinner = NacelleMassDecomposed(1,2);
    Mass_nozzle = sum(NacelleMassDecomposed(3,1:2));
    Mass_compartment = NacelleMassDecomposed(2,2);

    %                   Masslist=        [m_motor,Mass_fan,                Mass_fanframe,
Mass_shaft,Mass_inlet,Mass_fanCowl,Mass_spinner,   Mass_compartment,Mass_nozzle];
    Masslist=                  [Mass_fan,                                  Mass_fanframe,
Mass_shaft,Mass_inlet,Mass_fanCowl,Mass_spinner,   Mass_compartment,Mass_nozzle];
    %      Masslegend=['Motor',    'Fan','Fan Burst Frame', 'Shaft connection',    'Inlet',
'Fan Cowl',    'Spinner','Motor compartment','Nozzle'];
    Masslegend=['Fan','Fan  Burst  Frame',  'Shaft  connection',     'Inlet',    'Fan  Cowl',
'Spinner','Motor compartment','Nozzle'];

    pie(Masslist,Masslegend);
end

%Dimension of the engine (max radius section and length)
Dim_engine = [max(max(rduct)),max(max(xpos))];
```

FanMechMass

```
function [Mass,MassElements] = FanMechMass(rduct)
% Weight of the fan


% Program to compute the cruise fan weight
% This program is based on the adapatation of theNASA WATE engine weight formulation
% by Mark Waters.
% It is being used to compute the weight of a variable pitch fan w/Exit Guide Vane
% The input to this function should be in SI units. (all necessary
% conversion is performed internaly


Testingmode=0;      % should be 0 unless you are testing the function
%% For testing Make sure you comment the function line
if Testingmode~=0
    clc
    clear all
    close all

    %% Duct, Fan Design Variables
    A_fan = 1;                  % Fan Area
    rA_eMin = .0;               % Minimum exhaust area as a ratio of A_fan
    rA_eMax = .8;               % Maximum exhaust area as a ratio of A_fan
```

```
    rA_in = .95;                % Inlet throat area as a ratio of A_fan
    %           1     %2     %3     %4
    DVDuct=[A_fan;rA_in;rA_eMin;rA_eMax];

    MotorCompDim = [0.53,0.16]; % Length & radius of motor in [m]
    [xpos,rduct]=FanGeometry(DVDuct,MotorCompDim,0); % Geometry of the duct
    %including the fan tip and hub radii (in position 2,1 and 2,2
    %respectively)
end


DtipFan = rduct(2,1)*3.2808;     %First Stage Tip Diameter      [ft]
Utip2 = FanMap(1,2,10)*3.2808;   %Design 1st Stage Tip Speed    [ft/sec]
v_2 = FanMap(1,2,11)*3.2808;     %Face Axial Velocity           [ft/sec]

%(Stators include Exit Guide Vane)
Lshaft = 0.15*3.2808;            %Shaft length                 [ft]
HTratio = rduct(2,2)/rduct(2,1); %Hub-Tip Ratio

%% Constants
DiqDo = 0.8;                     %Shaft Inside Diameter / Outside Diameter
rho_mat = 290.3;                 %Material Density             [lb/ft3]
TR_cmp = 1;                      %Blade/Vane Taper Ratio
OSF = 1.05;                      %Over Speed Factor
ArRotor = 2.2;                   %Blade Aspect RAtio
CqSRotor = 0.9;                  %Blade Solidity
ARStator = 3.1;                  %Stator Aspect RAtio
CqSStator = 1.1;                 %Stator Solidity
NrowRot = 1;                     %Number or Rotor Rows
NrowSta = 2;                     %Number of Stator Rows

TRatio = TR_cmp;

Dtip = DtipFan * 12;
Dhub = Dtip * HTratio;
Dpitch = (Dtip + Dhub) / 2;
Span = (Dtip - Dhub) / 2;
Utip = Utip2;
Phi = v_2 / Utip;
M_2 = v_2/1116.5;                        %Face Mach No.

if (ArRotor < 0.78)
    ArRotor = 0.78;
end

if (ARStator < 1.18)
    ARStator = 1.18;
end

if (CqSRotor < 0.85)
    CqSRotor = 0.85;
end

if (CqSStator < 0.765)
    CqSStator = 0.765;
end

%Maximum Tip Speed -- Tip Speed * Over Speed Factor
UtipMax = Utip * OSF;

%Blade & Vane Volumes -- Equation 3 in WATE document NASA CR 159481
%in cubic inches
kvol = 0.05;           %This value recommended by WATE documentation
if (HTratio > 0.8)
    kvol = kvol + 0.04 * (HTratio - 0.8);
end
VolBlade = kvol * Span ^ 3 / ArRotor ^ 2;
VolStator = kvol * Span ^ 3 / ARStator ^ 2;

%Number of Rotor Blade & Number of Stator Vanes
Nblade = round(3.1416 * Dtip * CqSRotor * ArRotor / Span);
NVane = round(3.1416 * Dtip * CqSStator * ARStator / Span);

%Rotor Blades & Stator Vanes Weight, lb
WgtRotor = (VolBlade / 1728) * rho_mat * Nblade * NrowRot;
WgtStator = (VolStator) / 1728 * rho_mat * NVane * NrowSta;

%Rotor, Stator & Stage Lengths in inches
Lrotor = Span / ArRotor;
Lstator = Span / ARStator;
```

```
if (NrowSta == 0)
    Lstator = 0;
end
Lspace = 0.17 * Lrotor;
Lstage = Lrotor + Lstator + Lspace;

%Disk -- Pull Stress -- psi, Volume -- in3 & Weight -- lb
SigmaDsk = (12 * (rho_mat / 1728) * UtipMax ^ 2 / (32.174 * TRatio)) *...
    (((1 - HTratio ^ 2) / 2) + (TRatio - 1) * (1 - HTratio) * (1 + 3 * HTratio) / 12);

%  The following parameter is the Pull Stress*hub radius/e05
%  The parameter is used in Figure 9 in the WATE volume
sigmaRhub = SigmaDsk * (Dhub / 2) / 100000;

if (rho_mat < 300)%  Titanium -- Volume/HubDiameter^2
    VdskqDh2 = -0.0103 * sigmaRhub ^ 4 + 0.0399 * sigmaRhub ^ 3 + ...
        0.0527 * sigmaRhub ^ 2 - 0.0527 * sigmaRhub + 0.12;
    if (sigmaRhub < 0.7)
        VdskqDh2 = 0.12;
    elseif (sigmaRhub > 3)
        VdskqDh2 = 0.34 * sigmaRhub - 0.34;
    end
else %  Steel -- Volume/HubDiameter^2
    VdskqDh2 = -0.0048 * sigmaRhub ^ 4 + 0.0388 * sigmaRhub ^ 3 - ...
        0.0485 * sigmaRhub ^ 2 + 0.0189 * sigmaRhub + 0.1196;
    if (sigmaRhub < 0.7)
        VdskqDh2 = 0.12;
    elseif (sigmaRhub > 3)
        VdskqDh2 = 0.295 * sigmaRhub - 0.52;
    end
end

VolDsk = VdskqDh2 * Dhub ^ 2;
WgtDsk = (VolDsk / 1728) * rho_mat;

%Hardware Weight (tie rods, etc.)
WgtHW = 0.75 * 3.1416 * Dhub * 0.075 * Lstage * rho_mat / 1728;

%Case Weight
WgtCase = 3.1416 * Dtip * Lstage * 0.1 * rho_mat / 1728;

%Fan Stage Weight
Wgt_Stage = WgtRotor + WgtStator + WgtDsk + WgtHW + WgtCase;
% LFan = Lstage;

%% Final Calculations including Shaft W & L, Frame W, Overall W %%
WaFan = 222.6;                      %Fan Airflow [lb/sec]
h2 = 124.0;                         %Fan Entrance Enthalpy [Btu/lb]
h22 = 135.0;                        %Fan Exit Enthalpy [Btu/lb]
FanWork = WaFan * (h22 - h2);       %Power in [Btu/sec]

% Parametric Evaluation of Shaft Weight

Rtip_ft = Dtip / 2 / 12;
RotSpd_shaft = Utip2 / Rtip_ft;
Torque_shaft = 778 * FanWork / RotSpd_shaft;
Stress_allow = 115000;

%Assume shaft is steel -- Density = 490 lb/ft3
Dens_shaft = 490 / 1728;
Table = zeros(3,7);
DiqDo = 0.9;
for L=1:1:7
    Do_shaft = (16 * (12 * Torque_shaft) / (3.1416 * Stress_allow * ...
        (1 - DiqDo ^ 4))) ^ (1 / 3);
    Wgt_Shaft = Lshaft * Dens_shaft * 0.786 * Do_shaft ^ 2 * ...
        (1 - DiqDo ^ 2);

    Table(1,L) = DiqDo;
    Table(2,L) = Do_shaft;
    Table(3,L) = Wgt_Shaft;
    DiqDo = DiqDo - 0.1;        %Shaft Inside Diameter / Outside Diameter
end

Do_shaft = (16 * (12 * Torque_shaft) / (3.1416 * Stress_allow * ...
    (1 - DiqDo ^ 4))) ^ (1 / 3);
Wgt_Shaft = Lshaft * Dens_shaft * 0.786 * Do_shaft ^ 2 * (1 - DiqDo ^ 2);


%% Frame Weight
```

```
%% Gas Generator
Radius_feet = Dtip / 12 / 2;
Rad_SQ = Radius_feet ^ 2;

% Front Frame is TYPE 1 in the WATE document
% This is the fan frame of a turbofan engine -- there is no power take off
if (Rad_SQ < 3)
    Wgt_frame1 = 33.33 * Rad_SQ;
else
    Wgt_frame1 = 42 * Rad_SQ - 26;
end
%Assume the Frame is a Type 1 Front Frame

Wgt_Frame = Wgt_frame1;

% Preparing outputs
MassElements = [Wgt_Stage, Wgt_Shaft, Wgt_Frame];

% Total Weight = Stage + Shaft + Frame
Mass = Wgt_Stage + Wgt_Shaft + Wgt_Frame;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function mass = massductelement(x,r,P,Sigma,Ro)
Testingmode=0;      % should be 0 unless you are testing the function
%% for testing Make sure you comment the function line
if Testingmode~=0
    clc
    clear all
    close all

    %% Input data
    xi = 0.802120164*0.3048;    %[m]
    xe = 1.80996156*0.3048;     %[m]

    ri = 1.395975802*0.3048;    %[m]
    re = 1.519723166*0.3048;    %[m]

    Pi = 18.63*6894.757; %[Pa]
    Pe = 18.58*6894.757; %[Pa]

    % Yield strength
    Sigma = 11 * 10^6; %[Pa]
    % Density of material
    Ro =2.7 *10^-3 *10^(2*3); %[kg/m3]
    x = [xi,xe];
    r = [ri,re];
    P = [Pi,Pe];
end


%% Preparation of linear approximation
dPdx= (P(2)-P(1))/(x(2)-x(1));
drdx= (r(2)-r(1))/(x(2)-x(1));

Pxo = P(1)-x(1)*dPdx;
rxo = r(1)-x(1)*drdx;


% r(x) = drdx * x + rxo
% P(x) = dPdx * x + Pxo

% t(x) = P(x) * r(x) / Sigma;
% dm = Ro * 2 * pi * r(x) * t(x) * dx
% th(1)n
% dm = Ro * 2 * pi / Sigma * P(x) * r(x)^2 * dx

% By integrating the linear approximation:


mass = (dPdx*drdx * drdx)/4 *(x(2)^4-x(1)^4);
mass = mass + ((dPdx*rxo+drdx*Pxo) * drdx + dPdx*drdx * rxo)/3 * (x(2)^3-x(1)^3);
mass = mass + (Pxo*rxo * drdx + (dPdx*rxo+drdx*Pxo) * rxo)/2 * (x(2)^2-x(1)^2);
mass = mass + (Pxo*rxo * rxo)* (x(2)-x(1));
mass = Ro * 2 * pi / Sigma * mass;
```

FanPreSizing

```
function Afanmin = FanPreSizing(M_aK,AltK,FR_ThrustMax)

Testingmode=0;      % should be 0 unless you are testing the function
%% for testing Make sure you comment the function line
if Testingmode~=0
    clc
    clear all
    close all

    FR_ThrustMax = 36.2e3*[1,0.7,0.3 ,0.24, 0.3,0.7,1];
    FR_ThrustNom = 36.2e3*[1,0.7,0.25,0.22,0.25,0.7,1];
    Duration =            [5, 15,  30,  90,  30, 15,5];
    M_aK  = [.01,0.4,0.6,0.85,0.6,0.4,0.01];
    AltK = [0,15, 260, 350,  260, 15, 0]*100;
end

FanOpline = 2; Nmax=FanMap(0,0,6);
K=length(M_aK);
Akmin=zeros(K,1);

%-Pre-sizing phase 1
for k=1:K
    M_a = M_aK(k);Alt = AltK(k);
    ThrustTarget=FR_ThrustMax(k);
    Amin = 0.2;     %Minimum fan size
    Amax = 9;       %Maximum fan size
    Sacc = 0.001;MaxIt = 100; Abest = 0;it = 1;
    Atable = zeros(MaxIt,1); S = zeros(MaxIt,1);%ThrustT=S;
    Test = zeros(MaxIt,1);
    S(1) = (Amax - Amin)/5;   %Initial step
    Atable(1) = Amin;  %Initial guess

    %-02
    while (it<MaxIt)&&(S(it)>Sacc)&&(Abest~=Amax)&&(Amax>Amin)
        %-03
        Area = Atable(it); DVDuct =[Area,0.95,0,1];
        [NM,NM,NM,NM,Thrustelements]=FanModel(M_a,Alt,DVDuct,...
            [20,20],FanOpline,Nmax,0);
        Thrust=Thrustelements(1);
        if Thrust<ThrustTarget
            A2small=1;
        else
            A2small=0;
        end
        %-04
        if A2small==1
            %-08
            Test(it) = 1;
        else
            %-06
            if Abest==0 || Atable(it)<Abest
                %-07
                Abest = Atable(it);
            end
            %-09
            Test(it) = -1;
        end
        %-10
        if it~=1 && Test(it)*Test(it-1)<0
            %-11
            S(it+1) = S(it)/(1.61803399*2);
        else
            S(it+1) = S(it);
        end
        %-13
        if A2small==1
            %-15
            Atable(it+1) = min(Atable(it) + S(it+1),Amax);
        else
            %-14
            Atable(it+1) = max(Atable(it) - S(it+1),Amin);
        end
        %-16
        it=it+1;
    end
    %-17
```

```
    Akmin(k) = Abest; % Stores the minimum fan area which will allow
    % providing the thrust required
end

%-Pre-sizing phase 2
Afanmin = max(Akmin); % The minimum fan area corresponds to the smallest
% fan area which can perform all scenario
```

MotorPreSizing

```
function roStart=MotorPreSizing(M_aK,AltK,FR_ThrustMax,Afanmin,ARmDefault,Pplot)

Testingmode=0;      % should be 0 unless you are testing the function
%% for testing Make sure you comment the function line
if Testingmode~=0
    clc
    clear all
    close all

    FR_ThrustMax = 36.2e3*[1,0.7,0.3 ,0.24];
    M_aK  = [.01,0.4,0.6,0.85];
    AltK = [0,15, 260, 350]*100;
    Afanmin = 0.6573;
    ARmDefault = 4;
    Pplot = 1;
end


%% Determination of the fan power need (presizing phase 3)
DVDuct =[Afanmin,0.95,0,1];
FanOpline = 2; Nmax=FanMap(0,0,6);

K=length(M_aK);
N = zeros(K,1); Pow = zeros(K,1); Tor = zeros(K,1);Ncs=N;Nrad=N;
for k=1:K
    ThrustTarget = FR_ThrustMax(k);

[ShaftOp]=Thrustpoint(DVDuct,FanOpline,[.8,0.2],ThrustTarget,[Nmax,0,0],M_aK(k),AltK(k),0
);

    Ncs(k)=ShaftOp(1);
    N(k) = ShaftOp(2);        % Shaft speed in RPM
    Nrad(k) = N(k)*2*pi/60;      % Shaft speed in rad/sec
    Pow(k) = ShaftOp(3);      % Minimum power necessary to perform thrust requirement [W]
    Tor(k) =  Pow(k)/Nrad(k);    % Minimum torque necessary to perform thrust requirement
[Nm]
end
% Deriving the motor limits necessary to perform the mission
Nmin = max(N); PowMin = max(Pow); TorMin = max(Tor);


%% searching for the motor initial size (presizing phase 4)
rmin = 65;     %Minimum motor radius
rmax = 300;    %Maximum motor radius
Sacc = 0.001;MaxIt = 100; rbest = 0;it = 1;
rtable = zeros(MaxIt,1); S = zeros(MaxIt,1);%ThrustT=S;
Test = zeros(MaxIt,1);
S(1) = (rmax - rmin)/5;   %Initial step
rtable(1) = rmin;  %Initial guess
PowLim = Test;TorLim=Test;NLim=TorLim;
%-02
while (it<MaxIt)&&(S(it)>Sacc)&&(rbest~=rmax)&&(rmax>rmin)
    %-03
    ro = rtable(it); DVMotor =[300,ARmDefault,45.4,94,4,20,ro];
    [OpLim]= HSTlimits(DVMotor);
    PowLim(it) = OpLim(1);
    TorLim(it) = OpLim(2);
    NLim(it)= OpLim(3);

    if OpLim(1)<PowMin || OpLim(2)<TorMin || OpLim(3)<Nmin
        r2small=1; % the motor is too small
    else
        r2small=0; % the motor is ok (but maybe too large)
    end
    %-04
    if r2small==1
        %-08
        Test(it) = 1;
```

```
    else
        %-06
        if rbest==0 || rtable(it)<rbest
            %-07
            rbest = rtable(it);
        end

        %-09
        Test(it) = -1;
    end
    %-10
    if it~=1 && Test(it)*Test(it-1)<0
        %-11
        S(it+1) = S(it)/(1.61803399*2);
    else
        S(it+1) = S(it);
    end
    %-13
    if r2small==1
        %-15
        rtable(it+1) = min(rtable(it) + S(it+1),rmax);
    else
        %-14
        rtable(it+1) = max(rtable(it) - S(it+1),rmin);
    end
    %-16
    it=it+1;
end
%-17
roStart = rbest; % returns the minimum motor radius which will allow
% providing the thrust required

if Pplot~=0
    figure(Pplot)
    subplot(3,1,1)
    plot(rtable,PowLim,'*b')
    hold on
    plot([min(rtable),max(rtable)],[PowMin,PowMin],'-r')

    subplot(3,1,2)
    plot(rtable,TorLim,'*b')
    hold on
    plot([min(rtable),max(rtable)],[TorMin,TorMin],'-r')

    subplot(3,1,3)
    plot(rtable,NLim,'*b')
    hold on
    plot([min(rtable),max(rtable)],[Nmin,Nmin],'-r')
end
```

B.5.4 Functions dedicated to surrogate modeling

RunDoE script

```
clear all
clc
close all

%% Mission points
Alt = [0         50      275     350     150     100     100     0]*100;%[ft]
M_a  = [0.04     0.41   0.75    0.78    0.71    0.6     0.45    0.3];%[-]

np = 25;%44;
nRandom = floor((np^3)*0.0);

n2p = 4; % Number of pole pairs

%% Design variables Ranges
% DV(1) = Afan    % DV(2) = ro    % DV(3) = ARm
r=1;% A_fan - Fan Area [m2] - Typical range [0.2-9]
lb(r)= 0.2;     ub(r)= 9;   npt(r)=np;
```

```
r=2;% ro  - Mean armature radius  (mm) - Typical range [65-300] Nom 140
lb(r)= 65;      ub(r)= 320;   npt(r)=np;

r=3;% ARm - Machine shape factor (La/ro) - Typical range [1.6-8]
lb(r)= 1.6;       ub(r)= 8;   npt(r)=np;

nDV=r;          % Total number of design variables
K= length(Alt); % Total number of operational conditions to explore

%% Build full factorial Design of Experiement
Xlevels = zeros(nDV,max(npt));
for dv=1:nDV
    Xlevels(dv,:)=linspace(lb(dv),ub(dv),npt(dv));
end

X=zeros(3,prod(npt));

Nexp=0;%Number of experiment in the DOE
for dv1=1:npt(1)
    for dv2=1:npt(2)
        for dv3=1:npt(3)
            Nexp=Nexp+1;
            X(:,Nexp)=[Xlevels(1,dv1);Xlevels(2,dv2);Xlevels(3,dv3)];
        end
    end
end

%% Setup of random samples
if nRandom~=1
    Ndoe=Nexp;
    Nexp=(Nexp+nRandom)
    for n=Ndoe+1:Nexp
        for i=1:3
            X(i,n)=lb(i)+rand()*(ub(i)-lb(i));
        end
    end
end
c=clock;
disp(sprintf('      %d:%d:%d',c(4),c(5),floor(c(6))));
disp(sprintf('The estimated runtime for the experiement: %3.f minutes',Nexp*0.061/60));

%% Preparation of response matrices
ThCap=zeros(Nexp,K);
m_engine=zeros(Nexp,1);
EffCoe1=zeros(Nexp,K);
EffCoe2=zeros(Nexp,K);
EffCoe3=zeros(Nexp,K);
Progress=[0,0,0];
Centile=Nexp/100;

%% Get responses
for n=1:Nexp
    [ThCap(n,:),m_engine(n),EffCoen]=DuctedFanResponse1(X(:,n),Alt,M_a,n2p);
    EffCoe1(n,:)=EffCoen(1,:);
    EffCoe2(n,:)=EffCoen(2,:);
    EffCoe3(n,:)=EffCoen(3,:);
    Progress(2) = Progress(2) +1;
    if Progress(2)>Centile && Nexp>=800
        Progress(1)=Progress(1)+1;
        Progress(2)=0;
        disp(sprintf('%d%%  -  Time  to  completion:  %d  min',Progress(1),floor((Nexp-
n)*0.061/60)))
        if Progress(1)>=10
            if mod(Progress(1),10)==0
                c=clock;
                disp(sprintf('      %d:%d:%d',c(4),c(5),floor(c(6))));
            end
        end
    end
end
Labels=['Th1','Th2','Th3','Th4','Th5','Th6','Th7','Th8','m_engine',...

'EffCoef1_1','EffCoef1_2','EffCoef1_3','EffCoef1_4','EffCoef1_5','EffCoef1_6','EffCoef1_7
','EffCoef1_8',...

'EffCoef2_1','EffCoef2_2','EffCoef2_3','EffCoef2_4','EffCoef2_5','EffCoef2_6','EffCoef2_7
','EffCoef2_8',...

'EffCoef3_1','EffCoef3_2','EffCoef3_3','EffCoef3_4','EffCoef3_5','EffCoef3_6','EffCoef3_7
','EffCoef3_8',...
```

```
    'X1','X2','X3'];
ResultTable=[ThCap,m_engine,EffCoe1,EffCoe2,EffCoe3,X'];
save 'ElecFanDoE25.mat' ResultTable
csvwrite('ElecFanDoE25.csv',ResultTable)
```

DuctedFanResponse

```
function [ThCap,m_engine,EffCoe]=DuctedFanResponse(Xo,Alt,M_a,n2p)
Testingmode=0;      % should be 0 unless you are testing the function
%% for testing Make sure you comment the function line
if Testingmode~=0
    clc
    clear all
    close all

    %% Mission points
    Alt = [0         50      275     350     150     100     100     0]*100;%[ft]
    M_a  = [0.04     0.41   0.75    0.78    0.71    0.6     0.45    0.3];%[-]
    K= length(Alt);

    %% Design variables Ranges
    % DV(1) = Afan    % DV(2) = ro    % DV(3) = ARm
    r=1;% A_fan - Fan Area [m2] - Typical range [0.2-9]
    lb(r)= 0.2;     ub(r)= 9;
    Xo(r)= 1;

    r=2;% ro  - Mean armature radius  (mm) - Typical range [65-300] Nom 140
    Xo(r)= 170;     lb(r)= 65;      ub(r)= 400;

    r=3;% ARm - Machine shape factor (La/ro) - Typical range [1.6-8]
    Xo(r)= 4;       lb(r)= 1.6;       ub(r)= 8;

    n2p=4; %Number of pole pairs in motor
end

%% Setup DV for calculation
X = zeros(7,1);
r=1;X(r)= 300;          % Ks - Armature Ampere turn loading (kA/m)
r=2;X(r)= Xo(3);        % ARm - Machine shape factor (La/ro)
r=3;X(r) = 45.4;        % eis - Electrical radial airgap (mm) (5 mm is pretty small)
r=4;X(r) = 94;          % Ispe - Armature current density (Arms/mm2)
r=5;X(r)= 20;           % ToP  - Temperature (K)
r=6;X(r)= Xo(2);        % ro  - Mean armature radius  ro (mm)
r=7;X(r)= Xo(1);        % A_fan - Fan Area [m2]


DVMotor = [X(1),X(2),X(3),X(4),n2p,X(5),X(6)];
DVDuct =[X(7),0.95,0,1];
FanOpline = 2; %operating line chosen for the response
Nmin = FanMap(0,FanOpline,5); % Minimum corrected normalized speed for the fan
K= length(Alt); % Total number of operational conditions to explore

%% Evaluation of responses
[MotorLim,MotorDim] = HSTlimits(DVMotor);
MotorCompDim=MotorCompartementSizing(MotorDim);

% Initiation of the response variables
Th= zeros(2,K);% Stores thrust variables
% Th(1,:): Max thrust capability
% Th(2,:): Thrust at 50% thrust level
% Th(3,:): Min thrust (idle)
N = zeros(2,K);% Stores speed variables
% N(1,:): Max speed
% N(2,:): Thrust at 50% max speed
% N(3,:): Min speed (idle)
PowerProp=zeros(2,K);% propulsive power (same structure as above)

EPreq=zeros(2,K);% Electric power requirements

PoMaxStages=zeros(1,4); % Max total pressure in the duct
ShaftOp=zeros(2,3,K);
%ShaftOp(1,:,:)= Operation at full throttle (N normalized, N actual in RPM, Shaft Power
in [W])
%ShaftOp(2,:,:)= Operation at iddle (N normalized, N actual in RPM, Shaft Power in [W])

Pplot = 0;

%Evaluation of max speed and thrust
```

```
for k=1:K
    if Pplot==1

[Th(1,k),Nmax,Properties,PoStages,ShaftOpk]=MaxTrustCap3(DVDuct,MotorCompDim,MotorLim,M_a
(k),Alt(k),Pplot+k-1);
    else

[Th(1,k),Nmax,Properties,PoStages,ShaftOpk]=MaxTrustCap3(DVDuct,MotorCompDim,MotorLim,M_a
(k),Alt(k),0);
    end
    PowerProp(1,k)= Properties(11);
    ShaftOp(1,:,k)=ShaftOpk(:); %Operational characteristics of the shaft at full throtle
settings
    for st=1:length(PoMaxStages)
        PoMaxStages(st)=max(PoMaxStages(st),PoStages(st+1));
    end
end

DN4op = 0.05; % Minimum delta between Nmax and Nmin to say that the fan can operate at
the specified conditions

% Checks with the speeds that the engine is capable to operate in allscenarios, then
evaluate the thrust at
% thrust lever setting (speed): iddle, 50% and max
if min(ShaftOp(1,1,:))>Nmin+DN4op            % If the fan max speed is less that some
threshold at any scenario
    %it will be considered as a failure for all scenario
    N(1,:)=ShaftOp(1,1,:);
    N(2,:)=zeros(1,K)+Nmin;
    for k=1:K

[Th(2,k),PowerProp(2,k),ShaftOpk]=ThrustShaftOp(M_a(k),Alt(k),DVDuct,MotorCompDim,FanOpli
ne,N(2,k),0);
        ShaftOp(2,:,k)=ShaftOpk(:); %shaft operational characteristics of the  at minimum
speed
    end
    % Based on shaft power evaluate the ammount of electricity to provide
    [EPreq,m_motor]= HTSatt(MotorLim,ShaftOp,DVMotor);
    [m_engine,Dim_engine]=MassModel(DVDuct,MotorCompDim,PoMaxStages,m_motor,Pplot);

    %% Thrust capability
    ThCap=Th(1,:);  %Max thrust the engine is capable of producing in each scenario

    %% Determine the efficiency in each setting/at each scenario
    Eff=PowerProp./EPreq;
    EffCoe=zeros(3,K); %Quadratic coefficients for Eff=f(Th/Tcap)
    % Defined as:
    % -EffCoe(1,k) = Eta@max thrust;
    % -EffCoe(2,k) = Eta@idle thrust;
    % -EffCoe(3,k) = Idle thrust/Max thrust;
    for k=1:K
        EffCoe(1,k) = Eff(1,k);
        EffCoe(2,k) = Eff(2,k);
        EffCoe(3,k) = Th(2,k)/Th(1,k);
    end
else
    ThCap=zeros(1,K);
    m_engine=0;
    EffCoe=zeros(3,K);
end
```

FilterAndTrain.m

```
clc
clear all
close all
%% Mission,  DoE data , and settings
Alt = [0         50      275     350     150     100     100     0]*100;%[ft]
M_a  = [0.04     0.41   0.75    0.78    0.71    0.6     0.45    0.3];%[-]
%Load data
[Data,Labels,nlevels,rDV]=GetDoEData();
X=Data(:,rDV:rDV+2);% Matrix containing design variables
Ntot= length(X(:,1));%Number of point in the DoE
K = length(Alt); %total number of scenarios

% Training schedule
%for weight
TrainWeight=1;  % Set as 1 if the engine regression must be performed
```

```
%for others
Training=0; %1 to re-train nets / else to use previously trained nets

for  k=1:8 %This  routine  will  perform  the  NN  training  and  will  build  ML  files  that
correspond to each scenarios
dataCol=k;

%% Debugging setting (if not debugging all 0)
Pplot=0;
Ddisp=0;
PlotTesting=0;  % This  will  display  the  sorting  of  points  into  zones  (for  debugging
purposes only)

%% Organize
%Indexes
% a: index for fan area (Afan)
% r: index for motor radius (ro)
% s: index for shape factor (ARm)
BlankCell = 9999.99999;
Response = zeros(nlevels)+BlankCell;
Coef1 = zeros(nlevels);
Coef2 = zeros(nlevels);
Coef3 = zeros(nlevels);
ARm=X(1:nlevels(3),3);
Afan=zeros(nlevels(2),1);

for a=1:nlevels(2)
    rank=1+ (a-1)*nlevels(2)*nlevels(3);
    Afan(a)=X(rank,1);
end

ro=zeros(nlevels(1),1);
for r=1:nlevels(1)
    rank=1+ (r-1)*nlevels(3);
    ro(r)=X(rank,2);
end

for n=1:Ntot
    r=0;s=0;a=0;
    while a<nlevels(1)
        a=a+1;
        if X(n,1)==Afan(a)
            while r<nlevels(2)
                r=r+1;
                if X(n,2)==ro(r)
                    while s<nlevels(2)
                        s=s+1;
                        if X(n,3)==ARm(s)
                            Response(a,r,s)=Data(n,dataCol);
                            Coef1(a,r,s)=Data(n,dataCol+K+1);
                            Coef2(a,r,s)=Data(n,dataCol+2*K+1);
                            Coef3(a,r,s)=Data(n,dataCol+3*K+1);
                            if k==TrainWeight
                                Weight(a,r,s)=Data(n,K+1);
                            end
                            s=nlevels(3);
                        end
                    end
                    r=nlevels(2);
                end
            end
            a=nlevels(1);
        end
    end
end
for a=1:nlevels(1)%Verifies that no data is missing in the full factorial DoE
    for r=1:nlevels(2)
        for s=1:nlevels(3)
            if Response(a,r,s)==BlankCell
                fprintf('Error: the value for ARm=%0.2f, Afan=%0.2f and ro=%0.2f was not
found\n',ARm(s),Afan(a),ro(r))
                %%%%%%%TEMP
                Response(a,r,s)=500000;
                %END TEMP%%%%%%%%%
            end

        end
    end
end
```

```
%% Boundary 0/2
Boundary02r=zeros(Ntot,3); %Straight up list of points on the boundary
lr=0;
for s=1:length(ARm)
    for a=1:length(Afan)
        r=1;
        while Response(a,r,s)==0 && r<=nlevels(2)
            r=r+1;
        end
        if r>nlevels(2)
            fprintf('Error Boundary 0/12 for ARm %0.2f and Afan %0.2f, the 0/1-2 boundary
was never found.\n',ARm(s),Afan(a))
        else
            lr=lr+1;
            Boundary02r(lr,:)=[Afan(a),ro(r),ARm(s)];
        end
    end
end
Boundary02=Boundary02r(1:lr,:);
Boundary02_3d=zeros(nlevels(2),3,nlevels(3));
for s=1:nlevels(3)
    Nstart=(s-1)*nlevels(2)+1;
    Nend = s*nlevels(2);
    Boundary02_3d(:,:,s)=Boundary02(Nstart:Nend,:);
end
clear Boundary02r;

%% Boundary 1/2 + Data Zone 1
Boundary12r_3d=zeros(nlevels(2),3,nlevels(3));
Boundary12r=zeros(Ntot,3); %Straight up list of values on the boundary (including Th,
Afan, ro, ARm)
rk12=0; % rank in Boundary12r

%   Thc_dat=zeros(nlevels(3),3);Thc_zone1=zeros(nlevels(3),2);   %   Storage   for   thrust
capacity values
Th1=zeros(1,2);Coef1Z1=zeros(2,1);Coef2Z1=zeros(2,1);Coef3Z1=zeros(2,1);
dnegdr=10; % this value specifies the maximum change required for considering that the
value has significantly changed
for s=1:length(ARm)
    a=nlevels(1);r=nlevels(2);lr=0;
    % Test if the surface is of type 2 or 1
    if Response(a,r,s)>Response(a,r-1,s)
        if Ddisp==1
            fprintf('                For ARm = %0.2f the shape is of type 2\n',ARm(s))
        end
        while Response(a,r,s)<Response(a-1,r,s)&& r>1
            a=a-1;
        end
        if a==1
            fprintf('Error: No b2 for ARm=%0.2f\n',ARm(s))
        else
            lr=1;
            % b1 identified
            Boundary12r_3d(lr,:,s)=[Afan(a),ro(r),ARm(s)];
            rk12=rk12+1;Boundary12r(rk12,:)=[Afan(a),ro(r),ARm(s)];
            Th1(rk12,:)=[Afan(a),Response(a,r,s)];
            Coef1Z1(rk12,:)=Coef1(a,r,s);
            Coef2Z1(rk12,:)=Coef2(a,r,s);
            Coef3Z1(rk12,:)=Coef3(a,r,s);
        end
    else
        if Ddisp==1
            fprintf('                For ARm = %0.2f the shape is of type 1\n',ARm(s))
        end
        while Response(a,r,s)<=Response(a,r-1,s)+dnegdr && r>1
            r=r-1;
        end
        if r==1
            fprintf('Warning: No b1 for ARm=%0.2f\n',ARm(s))
        elseif Response(a,r-1,s)==0
            fprintf('No zone 2 for ARm=%0.2f\n',ARm(s))
        else
            lr=1;
            % b1 identified
            Boundary12r_3d(lr,:,s)=[Afan(a),ro(r),ARm(s)];
            rk12=rk12+1;Boundary12r(rk12,:)=[Afan(a),ro(r),ARm(s)];
            Th1(rk12,:)=[Afan(a),Response(a,r,s)];
            Coef1Z1(rk12,:)=Coef1(a,r,s);
            Coef2Z1(rk12,:)=Coef2(a,r,s);
            Coef3Z1(rk12,:)=Coef3(a,r,s);
```

```
        end
    end
    % Evaluation of thrust equation (bs point)
    if a-1<1
        fprintf('Error: bs can not be estimated for ARm=%0.2f\n',ARm(s))
        %AlarmbsPt=1;
    else
        %AlarmbsPt=0;
    end
    % We now stand on the corner b
    while a>1
        a=a-1;
        while Response(a,r,s)<=Response(a,r-1,s)+dnegdr && r>1
            r=r-1;
        end
        if r==1
            % Absence of zone 0 and 2 in small value of ro (this condition should not
happend
            fprintf('Warning: Boundary touched ro-axis for ARm=%0.2f\n',ARm(s))
        else
            lr=lr+1;
            % Boundary identified
            Boundary12r_3d(lr,:,s)=[Afan(a),ro(r),ARm(s)];
            rk12=rk12+1;Boundary12r(rk12,:)=[Afan(a),ro(r),ARm(s)];
            Th1(rk12,:)=[Afan(a),Response(a,r,s)];
            Coef1Z1(rk12,:)=Coef1(a,r,s);
            Coef2Z1(rk12,:)=Coef2(a,r,s);
            Coef3Z1(rk12,:)=Coef3(a,r,s);
        end
    end
    if Pplot==1

        %Drawing
        n=s;
        Nstart=(n-1)*nlevels(2)+1;
        Nend = n*nlevels(2);
        figure(1)
        plot(Boundary02(Nstart:Nend,1),Boundary02(Nstart:Nend,2))
        hold on
        plot(Boundary12r_3d(:,1,n),Boundary12r_3d(:,2,n),'r')
    end
end

% Calculating the thrust/fan area gradient in zone 1
Zone1Alpha=mean(Th1(:,2)./Th1(:,1));
Zone1AlphaStore(k)=Zone1Alpha;
Zone1EffM=median(Coef1Z1);
Zone1EffMStore(k)=Zone1EffM;
Eff_idle=median(Coef2Z1);
Eff_idleStore(k)=Eff_idle;
Zone1TRmin=median(Coef3Z1);
Zone1TRminStore(k)=Zone1TRmin;
clear Coef1Z1;clear Coef2Z1;clear Coef3Z1;clear AlarmbsPt
Boundary12=Boundary12r(1:rk12,:);
clear Boundary12r

%% Drawing
% plot the thrust equations for all ARm values
if Pplot ==1
    figure(2)
    plot3(Boundary12(:,1),Boundary12(:,2),Boundary12(:,3),'.')
end

if Pplot>=1
    n=25;
    Nstart=(n-1)*nlevels(2)+1;
    Nend = n*nlevels(2);
    figure(3)
    plot(Boundary02(Nstart:Nend,1),Boundary02(Nstart:Nend,2))
    hold on
    Boundary12T=Boundary12r_3d(:,:,n);
    plot(Boundary12r_3d(:,1,n),Boundary12r_3d(:,2,n),'r')
end
clear Bounadary12T;clear Nend;clear Nstart;

%% Classification of points in zones

% Classification of sample points by zones
lr=zeros(3,1);rW=0;
% ZoneOpointT=zeros(Ntot,4);
```

```
Zone2pointT=zeros(Ntot,4);Zone2pointC1T=zeros(Ntot,4);Zone2pointC2T=zeros(Ntot,4);Zone2po
intC3T=zeros(Ntot,4);
for s=1:nlevels(3)
    if PlotTesting==1
        n=s;
        figure(4)
        Boundary02T=Boundary02_3d(:,:,s);
        plot(Boundary02T(:,1),Boundary02T(:,2))
        hold on
        Boundary12T=Boundary12r_3d(:,:,n);
        plot(Boundary12r_3d(:,1,s),Boundary12r_3d(:,2,s),'r')
        clear Boundary12T;
    end
    for a=1:nlevels(1)
        % definition of the 0/2 limit
        rank=1;Test=0;
        while Test==0
            if Afan(a)~=Boundary02_3d(rank,1,s)
                if rank<nlevels(1)
                    rank=rank+1;
                else
                    Test = 2;
                    fprintf('Error: Boundary02 is open\n')
                end
            else
                Test=1;
                rlim02 = Boundary02_3d(rank,2,s);
            end
        end
        % definition of the 1/2 limit
        rank =1;Test=0;
        while Test==0;
            if Afan(a)==Boundary12r_3d(rank,1,s)
                Test=1;rlim12 = Boundary12r_3d(rank,2,s);
            elseif rank == nlevels(1)
                Test=2;rlim12=max(ro);
            else
                rank=rank+1;
            end
        end
        if PlotTesting==1
            plot(Afan(a),rlim02,'*c')
            plot(Afan(a),rlim12,'*c')
        end
        % Identification of the zone location
        for r=1:nlevels(2)
            if PlotTesting==1
                plot(Afan(a),ro(r),'*g')
            end
            if rlim02> ro(r)   % this design is in zone 0
                lr(3)=lr(3)+1;
                if PlotTesting==1
                    plot(Afan(a),ro(r),'*k')
                end
            else % This engine is NOT in zone 0
                if k==TrainWeight
                    rW=rW+1;
                    WeightTraining(rW,:)=[Afan(a),ro(r),ARm(s),Weight(a,r,s)];
                end
                if   ro(r)<=rlim12 % this design is in zone2
                    lr(2) = lr(2)+1;
                    Zone2pointT(lr(2),:)=[Afan(a),ro(r),ARm(s),Response(a,r,s)];
                    Zone2pointC1T(lr(2),:)=[Afan(a),ro(r),ARm(s),Coef1(a,r,s)];
                    Zone2pointC2T(lr(2),:)=[Afan(a),ro(r),ARm(s),Coef2(a,r,s)];
                    Zone2pointC3T(lr(2),:)=[Afan(a),ro(r),ARm(s),Coef3(a,r,s)];
                    if PlotTesting==1
                        plot(Afan(a),ro(r),'*b')
                    end
                else    %Zone1
                    lr(1)=lr(1)+1;
                    if PlotTesting==1
                        plot(Afan(a),ro(r),'*r')
                    end
                end
            end
        end
    end
end
% Reducing matrices to non zero values
Zone2point=Zone2pointT(1:lr(2),:);Zone2pointC1=Zone2pointC1T(1:lr(2),:);
```

```
Zone2pointC2=Zone2pointC2T(1:lr(2),:);Zone2pointC3=Zone2pointC3T(1:lr(2),:);

clear Zone2pointT;clear Zone2pointC1T;clear Zone2pointC2T;clear Zone2pointC3T
clear Test; clear lr; clear r;clear a; clear s; clear n; clear rank; clear rk12;clear
rlimO2; clear rlim12;clear rW

%% Formating data points and fitting nets
if Training == 1
    Cellnets=cell(5,K);
    if k==1% Fitting boundary between zone 0 and 1 (Note: this limit is the same for all
scenarios, doing it once is sufficient)
        Temp = BoundaryO2(:,2);
        BoundaryO2(:,2)=BoundaryO2(:,3);
        BoundaryO2(:,3)=Temp;
        BndO2Labels=[cell2mat(Labels(rDV)),cell2mat(Labels(rDV+2)),'rO2'];
        clear Temp
        Net_bndO2 = Netfit(BoundaryO2,BndO2Labels,2,10); %%%%%%%Fitting nets
        Cellnets(5,1)=[Net_bndO2];
    end
    % Fitting boundary between zone 2 and 1
    Temp = Boundary12(:,2);
    Boundary12(:,2)=Boundary12(:,3);
    Boundary12(:,3)=Temp;
    Bnd12Labels=[cell2mat(Labels(rDV)),cell2mat(Labels(rDV+2)),'r12'];
    clear Temp
    Net_bnd12 = Netfit(Boundary12,Bnd12Labels,2,10);%%%%%%%Fitting nets
    Cellnets2(1,k)=[Net_bnd12];

    % Fitting thrust capacity in zone 2

Zone2Labels=[cell2mat(Labels(rDV)),cell2mat(Labels(rDV+1)),cell2mat(Labels(rDV+2)),cell2m
at(Labels(dataCol))];
    Net_Zone2 = Netfit(Zone2point,Zone2Labels,3,10);%%%%%%%Fitting nets
    Cellnets(1,k)=[Net_Zone2];

    %       Fitting       Eff       at       max       thrust       in       zone       2
Zone2EffMLabels=[cell2mat(Labels(rDV)),cell2mat(Labels(rDV+1)),cell2mat(Labels(rDV+2)),ce
ll2mat(Labels(dataCol+K+1))];
    Net_EffMZone2 = Netfit(Zone2pointC1,Zone2EffMLabels,3,25);%%%%%%%Fitting nets
    Cellnets(3,k)=[Net_EffMZone2];
    %              Fitting             TR             in             zone             2
Zone2TRLabels=[cell2mat(Labels(rDV)),cell2mat(Labels(rDV+1)),cell2mat(Labels(rDV+2)),cell
2mat(Labels(dataCol+3*K+1))];
    Net_TRZone2 = Netfit(Zone2pointC3,Zone2TRLabels,3,25);%%%%%%%Fitting nets
    Cellnets(4,k)=[Net_TRZone2];
    if                  k==TrainWeight%                  Fitting                  weight
WeightLabels=[cell2mat(Labels(rDV)),cell2mat(Labels(rDV+1)),cell2mat(Labels(rDV+2)),cell2
mat(Labels(K+1))];
        Net_Weight = Netfit(WeightTraining,WeightLabels,3,10);%%%%%%%Fitting nets
        Cellnets(5,2)=[Net_Weight];
    end
else
    load Cellnets
    if k==1
        Net_bndO2=cell2mat(Cellnets(5,1));
    end
    Net_bnd12=cell2mat(Cellnets(1,k));
    Net_Zone2=cell2mat(Cellnets(2,k));
    Net_EffMZone2=cell2mat(Cellnets(3,k));
    Net_TRZone2=cell2mat(Cellnets(4,k));

    if k==TrainWeight% Fitting weight
        Net_Weight=cell2mat(Cellnets(5,2));
    end
end
end%scanning scenarios
save 'Cellnets.mat' Cellnets Zone1AlphaStore Eff_idleStore Zone1EffMStore Zone1TRminStore
```

Netfit

```
function [Netformula]=Netfit(Data,Labels,nDV,Nnodes)
%% Inputs:
% Data: This is a matrix were the first nDV columns contain the design
% (i.e. independent) variables. The last column is the response (i.e.
% dependent) variable.
% Labels: Cell array providing the names of the variables stored in each
% column. ex: ['a','b','R']
% nDV: is the number of design variables (i,e, independent variables).
```

```
%
%% Output:
% Netformula: This is a string of character. This strings is a matlab
% formulation of the neural net returned by BRAINN. e.g.: R =
% exp(1-3*a)+3b

Testingmode=0;      % should be 0 unless you are testing the function
%% for testing Make sure you comment the function line
if Testingmode~=0
    % This function cannot be tested directly from within. Copy/paste this to an external
script to test it.
    % Otherwise the function "inputname" won't work.
    close all
    clc
    clear all

    Dater1234wruyh = zeros(9,3)-1;
    Labels = ['X1','X2','r'];

    x=Netfit2(Dater1234wruyh,Labels,2,5);
end

%% This sets up the xlsfile necessary to the execution of BRAINN
Root = sprintf('%s',inputname(1));% This uggly looking fellow returns the name of the
variables which was placed as input to the function
InputFileName=[Root,'.xls'];

A=size(Data);
if A(2)==3
    RangeL='A1:C1';
    RangeD=['A2:C',num2str(A(1)+1)];
elseif A(2)==4
    RangeL='A1:D1';
    RangeD=['A2:D',num2str(A(1)+1)];
elseif A(2)==5
    RangeL='A1:E1';
    RangeD=['A2:E',num2str(A(1)+1)];
else
    disp('Error the number DV is not prepared for')
end
delete(InputFileName)

xlswrite(InputFileName,Labels,RangeL) %Writting in the labels
xlswrite(InputFileName,Data,RangeD) %Writting in the data

rstruct.filename = InputFileName;
rstruct.num_in_var = nDV;% # of DV
rstruct.percent_val = 15;% percentage retained for validation
rstruct.percent_test = 0;%
rstruct.ran_val = 1; % rand for 1 and 0 for those at the end
rstruct.discrete = 0;

% read in the data
[num_responses] = data_reader(rstruct);

% Define inputs for neural network creation
nstruct.response_count = 1;% # of responses
nstruct.num_epochs = 200;% max # of interation
nstruct.train_time = 9000;% max run time [s]
nstruct.init_nodes = Nnodes;% structure # nodes to begin
nstruct.incr_nodes = 1;
nstruct.fin_nodes = Nnodes;%final
nstruct.iter_node = 1;
nstruct.graph = 1;% to display training graph
nstruct.epoch_update = 1;
nstruct.reg = 0;
nstruct.reg_param = 0;
nstruct.stop = 0;
nstruct.train_switch = 'trainbr';
nstruct.elim_outliers = 0;
nstruct.out_per = 10;
nstruct.memory_factor = 2;   %%%%% Factor to reduce memory load (1=normal, 2=half normal,
etc)
nstruct.JMP_format = 0;
nstruct.Excel_format = 0;
nstruct.MAT_format = 1;
nstruct.error_def = 2;     % 0 = % error, 1 = absolute, 2 = % error relative to range

nstruct.threshold = 0;   % Check against threshold to quit training early
nstruct.R2_train_thresh = 0;
```

```
nstruct.R2_val_thresh = 0;
nstruct.R2_test_thresh = 0;
nstruct.MFE_mean_thresh = 1;
nstruct.MFE_std_thresh = 1;
nstruct.MRE_mean_thresh = 1;
nstruct.MRE_std_thresh = 1;
nstruct.save_train_figure = 0;
nstruct.save_results_figure = 1;
nstruct.cont_train = 0;
nn_run(nstruct,1)
formulaTextfile=[cell2mat(Labels(nDV+1)),'_MAT.txt'];

% This command line imports the net's formula in a string of character
Netformula = fread(fopen(formulaTextfile,'r'),'*char')';

% Cleanup crew!!!! (so that the Matlab folder does not look like Dodd
% stadium after a game)
fclose('all');
delete(formulaTextfile)
delete(InputFileName)
delete 'NN_data.mat'
delete 'net_best.mat'
delete([cell2mat(Labels(nDV+1)),'_figure.jpg'])
delete([cell2mat(Labels(nDV+1)),'_stats.txt'])
clc
```

CodeGeneration

```
clc
clear all
close all

load Cellnets

%Mission
Alt = [0         50      275     350     150     100     100     0]*100;%[ft]
M_a  = [0.04     0.41   0.75    0.78    0.71    0.6     0.45    0.3];%[-]

% Transition settings
B_trans =0.4;   % Band overwhich the transition occurs
B_power = 5;    % Exponential shape of the transition
K=length(Alt);

c = clock;
Time=[num2str(c(3)),'/',num2str(c(2)),'/',num2str(c(1)),'                                    at
',num2str(c(4)),':',num2str(c(5)),':',num2str(floor(c(6)))];
Offset='    ';

for k=1:K
%% importing nets and data
Scen = num2str(k);  %flight condition number in string format
CRank = ['(',Scen,')'];
Zone1Alpha=Zone1AlphaStore(k);
Eff_idle=Eff_idleStore(k);
Zone1EffM=Zone1EffMStore(k);
Zone1TRmin=Zone1TRminStore(k);
if k==1
    Net_bnd02=cell2mat(Cellnets(5,1));
end
Net_bnd12=cell2mat(Cellnets(1,k));
Net_Zone2=cell2mat(Cellnets(2,k));
Net_EffMZone2=cell2mat(Cellnets(3,k));
Net_TRZone2=cell2mat(Cellnets(4,k));
if k==1% Fitting weight (you need to do this once only
    Net_Weight=cell2mat(Cellnets(5,2));
end

%% Pre-sizing functions
if k==1
    Signature=['%%  This  file  was  automatically  generated.\n%%  Model  and  regression
performed by Cyril de Tenorio\n',...
        '%% Fall 2009 - Georgia Institute of Technology\n','%% ',Time,'\n'];
    PreSizingMotor=['function
[roStart]=MotorPreSizing_reg(Afanmin,ARmDefault,Pplot)\n',Signature,...
        '%% This function will pre-size the motor of the electric ducted fan. It will do
so by picking a size of motor which will allow the fan (which is\n%% capable to perform
all requirements) to operate without ever encountering a physical limit for the motor
(infinitly large motor assumption).\n%%%% Input:\n%% Afanmin: Minimum fan area necessary
```

```
to perform the mission [m2]\n%% ARmDefault: Aspect ratio of the machine assumed for the 1st iteration [-]\n%% Pplot: if different than 0 the function will display the ro necessary in each condition\n%%%% Output:\n%% roStart= smallest motor radius which will allow the engine to behave as if it was powered by an infinitly large motor [mm].\n\nTestingmode=0;      %% should be 0 unless you are testing the function\n%%%% for testing Make sure you comment the function line\nif Testingmode~=0\n    clc\n    clear all\n    close all\n %%%% Mission\n    Afanmin = 0.55;\n    ARmDefault = 4;\n    Pplot = 1;\nend\n\n%% In order for the motor to behave as an inifinitly large motor, the fan must be saturated before the motor. \n %% It implies that the point (Afanmin,roStart,ARmDefault) must be in zone 1.\nAfan=Afanmin;\nARm=ARmDefault;\nOversize=1.3; %% This will oversize the motor by 10%% (Based on regression for the 12 boundary)\n\n'];
    PreSizingFan =['function Afanmin = FanPreSizing_reg(FR_ThrustMax,Pplot)\n',Signature,...
        '%% This function will pre-size the fan/duct of the electric ducted fan. It will do so by assuming that the fan is operated by an infinitly\n%% large motor.  Therefore this routine ignore the constraints from the motor and focuses on the fan.\n%%%% Input:\n%% FR_ThrustMax: Thrust requirement in each of the typical flight conditions\n%% Pplot: if different than 0 the function will display the thrust constraints and Afan necessary to meet them in each condition\n%%%% Output:\n%% Afanmin: Minimum fan area necessary to perform the mission [m2]\n\nTestingmode=0;      %% should be 0 unless you are testing the function\n%%%% for testing Make sure you comment the function line\nif Testingmode~=0\n    clc\n    clear all\n    close all\n    Pplot=1;\n %%%% Mission\n %%                    Pk   St   Taxi    Takeoff Climb1  Climb2  Cruize  Dsct1   Hold  Dsct2   Land.\n    Alt_t = [0          50      275     350     150     100     100     0]*100;%%[ft]\n    M_a_t  = [0.04       0.41    0.75    0.78    0.71    0.6     0.45    0.3];%%[-]\n    FR_ThrustMax = [    236     180     80      54      125     130     138     236]*1000/8;%%[N]\nend\nNc=length(FR_ThrustMax);\nZone1alpha=['];
end
PreSizingMotor=[PreSizingMotor,'c=',Scen,';\n',...
    Offset,'ro_12=',Net_bnd12,';\n',...
    Offset,'roInf(c)=Oversize*ro_12;\n'];
PreSizingFan=[PreSizingFan,num2str(Zone1Alpha)];
if k~=K
    PreSizingFan=[PreSizingFan,','];
else
    PreSizingMotor=[PreSizingMotor,'\nroStart=max(roInf);\nif Pplot~=0\n\n    figure(2)\nhold  on\n          for  c=1:8\n                    plot([c,c],[0,roInf(c)],''-r'')\nplot(c,roInf(c),''*r'')\n    end\nend\n'];
    fid = fopen('MotorPreSizing_reg.m','wt');
    fprintf(fid,PreSizingMotor);
    fclose(fid);
    PreSizingFan=[PreSizingFan,'];\n\n%%-Pre-sizing         phase1\nfor         c=1:Nc\nAcmin(c)=FR_ThrustMax(c)/Zone1alpha(c);  %%Minimum area for conditions c\n        if Pplot==1\n            hold on;\n            plot([0,2],[0,Zone1alpha(c)*2],''b'')\nplot([0,2],[FR_ThrustMax(c),FR_ThrustMax(c)],''r'')\nplot(Acmin(c),FR_ThrustMax(c),''r*'')\n            axis([0,1,0,max(FR_ThrustMax)*1.1])\nend\nend\n\n%%-Pre-sizing phase 2\nAfanmin = max(Acmin); %% The minimum fan area corresponds to the smallest\nif Pplot==1\nplot([Afanmin,Afanmin],[0,max(FR_ThrustMax)*1.1],''g'')\naxis([0,1,0,max(FR_ThrustMax)*1.1])\nend\n'];
    fid = fopen('FanPreSizing_reg.m','wt');
    fprintf(fid,PreSizingFan);
    fclose(fid);
end

%% Generating Thrust capability functions
if k==1 %Initialization of the file
    ThrustCapFctFile=['function ThCap=ApproxThrustCap(Afan,ro,ARm)\n',...
        '%% This file was automatically generated.\n%% Model and regression performed by Cyril de Tenorio\n',...
        '%% Fall 2009 - Georgia Institute of Technology\n','%% ',Time,'\n',...
        '%%%% Inputs:\n%%-Afan: Fan face area [m2]\n%%-ro: Motor rotor outer radius [mm]\n%%-ARm: machine aspect ratio [-]\n',...
        '%%%%  Output:\n%%-  ThCap:  Max thrust capability at typical flight conditions\n\n\n',...
        'ThCap=zeros(1,',num2str(K),');\n\n'];
end

ResponseT = 'ThCap';
FlightConditions = ['FL: ',num2str(Alt(k)/100),' and MN: ',num2str(M_a(k))];
MainFct=['%%%% Representation of Max thrust capability at ',FlightConditions,'\n',...
    'r02 =',Net_bnd02,';\n',...
    'if ro<r02\n',...
    Offset,ResponseT,CRank,'=0;\n',...
    'else\n',...
    'r12 = ',Net_bnd12,';\n'];

%Evaluation of transition
MainFct =[MainFct,'%% Settings used for transition\n',...
```

```
    Offset,'B_trans = ',num2str(B_trans),';\n',...
    Offset,'B_power = ',num2str(B_power),';\n',...
    Offset,'rt = r12-B_trans*(r12-rO2);%% Transition radius\n',...
    Offset,'if ro<rt\n',...
    Offset,Offset,'Beta=0;\n',...
    Offset,'elseif ro<r12\n',...
    Offset,Offset,'Beta=((ro-rt)/(r12-rt))^B_power;\n',...
    Offset,'else\n',...
    Offset, Offset,'Beta=1;\n',...
    Offset,'end\n'];
%Evaluation of response using transition
MainFct =[MainFct,...
    Offset,ResponseT,'Z1 = Afan *',num2str(Zone1Alpha,8),';\n',...
    Offset,'if Beta<1\n',...
    Offset,Offset,ResponseT,'Z2 = ',Net_Zone2,';\n',...
    Offset,Offset,'if ',ResponseT,'Z1 > ',ResponseT,'Z2\n',...
    Offset,Offset,Offset,ResponseT,CRank,'=       Beta*',ResponseT,'Z1       +       (1-
Beta)*',ResponseT,'Z2;\n',...
    Offset,Offset,'else\n',...
    Offset,Offset,Offset,ResponseT,CRank,'=',ResponseT,'Z1;\n',...
    Offset,Offset,'end\n',...
    Offset,'else\n',...
    Offset,Offset,ResponseT,CRank,'=',ResponseT,'Z1;\n',...
    Offset,'end\n',...
    'end\n'];
% Increment function code
ThrustCapFctFile=[ThrustCapFctFile,MainFct,'\n\n'];

%% Generating efficiency function
Response1 = 'Eff'; % Overall response
Response2 = 'EffMax';
Response3 = 'TR';
Scen = num2str(k);  %flight condition number in string format
if k==1
    Offset='    ';
    EffFctInit=['function          [',Response1,',',Response2,',',Response3,']          =
ApproxEff(Afan,ro,ARm,Treq,Tcap,Cond)\n',...
        '%% This file was automatically generated.\n%% Model and regression performed by
Cyril de Tenorio\n',...
        '%%  Fall  2009  -  Georgia  Institute  of  Technology\n','%%  ',Time,'\n%%%%
Inputs:\n',...
        '%%-Afan:  Fan  face  area  [m2]\n%%-ro:  Motor  rotor  outer  radius  [mm]\n%%-ARm:
machine aspect ratio [-]\n',...
        '%%-Treq: Thrust required [N]\n%%-Tcap: Max Thrust capacity in the given flight
condition [N]\n%%%% Output:\n',...
        '%%-',Response1,': Efficiency of the electric ducted fan providing Treq N of
thrust at Flight Conditions "Cond".\n',...
        '%% The flight conditions are defined as follows:\n',...
        '%% Cond         Altitude        MachNumber\n'];
    Eff_idleDef='\n\nEff_idleTable = [';
    EffMainFct = ['TR = ApproxTR(Afan,ro,ARm,Cond);   %% ratio idle/max thrust\n',...
        'Eff_idle = Eff_idleTable(Cond);   %% efficiency of fan at idle\n',...
        'TRreq=Treq/Tcap;                          %% thrust level at which the fan is
operating\n',...
        'if TRreq<TR %% The fan is operating under idle speed\n',...
        '    ',Response1,' = Eff_idle;\n',...
        '    EffMax=0;\n',...
        'else\n',...
        '    EffMax=ApproxEffMax(Afan,ro,ARm,Cond);\n',...
        '    ',Response1,' = Eff_idle+(EffMax-Eff_idle)/(1-TR)*(TRreq-TR);\n',...
        'end\n\n\n'];
end
% Listing of scenario at the beginning of the function
EffFctInit=[EffFctInit,...
     '%% ',Scen,'          ',num2str(Alt(k)),'          ',num2str(M_a(k))];

% Contributes to the listing of the idle efficiency value
if k<K
    Eff_idleDef=[Eff_idleDef,num2str(Eff_idle),';'];
else
    Eff_idleDef=[Eff_idleDef,num2str(Eff_idle),'];\n\n'];
end

% EffMax subfunction
Scen = num2str(k);  %flight condition number in string format
Response2 = 'EffMax';
CRank = ['(',Scen,')'];
if k==1
    EffMaxSubFct           =           ['function           ',Response2,'           =
Approx',Response2,'(Afan,ro,ARm,Cond)\n',...
```

```
        '%%Output:',Response2,'Overall efficiency at max thrust regime [-]\n\n'];
    EffMaxSubFct=[EffMaxSubFct,'if Cond==1\n'];
else
    EffMaxSubFct=[EffMaxSubFct,'elseif Cond==',Scen,'\n'];
end

EffMaxSubFct=[EffMaxSubFct,...
    '%%%% Representation of Max thrust efficiency at ',FlightConditions,'\n',...
    Offset,'rO2 = ',Net_bndO2,';\n',...
    Offset,'if ro<rO2\n',...
    Offset,'    ',Response2,' = 0;\n',Offset,'else\n',...
    Offset,'     r12 = ',Net_bnd12,...
    Offset,';\n',...
    Offset,'    %% Settings used for transition\n',...
    Offset,'    B_trans = ',num2str(B_trans),';\n',...
    Offset,'    B_power = ',num2str(B_power),';\n',...
    Offset,'    rt = r12-B_trans*(r12-rO2);%% Transition radius\n',...
    Offset,'    if ro<rt\n',...
    Offset,'        Beta=0;\n',...
    Offset,'    elseif ro<r12\n',...
    Offset,'        Beta=((ro-rt)/(r12-rt))^B_power;\n',...
    Offset,'    else\n',...
    Offset,'        Beta=1;\n',...
    Offset,'    end\n',...
    Offset,'    ',Response2,Scen,'Z1 = ',num2str(Zone1EffM),';\n',...
    Offset,'    if Beta<1    %% The design is in zone 2\n',...
    Offset,'        ',Response2,Scen,'Z2 = ', Net_EffMZone2,';\n',...
    Offset,'        if ',Response2,Scen,'Z2 > ',Response2,Scen,'Z1\n',...
    Offset,'                            ',Response2,'  =  Beta*',Response2,Scen,'Z1  +  (1-
Beta)*',Response2,Scen,'Z2;\n',...
    Offset,'        else    %% The design is in zone 1\n',...
    Offset,'            ',Response2,'=',Response2,Scen,'Z1;\n',...
    Offset,'        end\n',Offset,'    else\n',...
    Offset,'                                ',Response2,'=',Response2,Scen,'Z1;\n',Offset,'
end\n',Offset,'end\n'];
if k==K
    EffMaxSubFct=[EffMaxSubFct,'end\n\n\n'];
end

% TR subfunction
if k==1
    TRSubFct = ['function ',Response3,' = Approx',Response3,'(Afan,ro,ARm,Cond)\n',...
        '%%Output:',Response3,'Idle to max thrust ratio [-]\n'];
    TRSubFct=[TRSubFct,'if Cond==1\n'];
else
    TRSubFct=[TRSubFct,'elseif Cond==',Scen,'\n'];
end
TRSubFct=[TRSubFct,...
    '%%%% Representation of Max thrust efficiency at ',FlightConditions,'\n',...
    Offset,'rO2 = ',Net_bndO2,';\n',...
    Offset,'if ro<rO2\n',...
    Offset,'    ',Response3,' = 0;\n',Offset,'else\n',...
    Offset,'     r12 = ',Net_bnd12,...
    Offset,';\n',...
    Offset,'    %% Settings used for transition\n',...
    Offset,'    B_trans = ',num2str(B_trans),';\n',...
    Offset,'    B_power = ',num2str(B_power),';\n',...
    Offset,'    rt = r12-B_trans*(r12-rO2);%% Transition radius\n',...
    Offset,'    if ro<rt\n',...
    Offset,'        Beta=0;\n',...
    Offset,'    elseif ro<r12\n',...
    Offset,'        Beta=((ro-rt)/(r12-rt))^B_power;\n',...
    Offset,'    else\n',...
    Offset,'        Beta=1;\n',...
    Offset,'    end\n',...
    Offset,'    ',Response3,Scen,'Z1 = ',num2str(Zone1TRmin),';\n',...
    Offset,'    if Beta<1    %% The design is in zone 2\n',...
    Offset,'        ',Response3,Scen,'Z2 = ', Net_TRZone2,';\n',...
    Offset,'        if ',Response3,Scen,'Z2 > ',Response3,Scen,'Z1\n',...
    Offset,'                            ',Response3,'  =  Beta*',Response3,Scen,'Z1  +  (1-
Beta)*',Response3,Scen,'Z2;\n',...
    Offset,'        else    %% The design is in zone 1\n',...
    Offset,'            ',Response3,'=',Response3,Scen,'Z1;\n',...
    Offset,'        end\n',Offset,'    else\n',...
    Offset,'                                ',Response3,'=',Response3,Scen,'Z1;\n',Offset,'
end\n',Offset,'end\n\n'];
if k==K
    TRSubFct=[TRSubFct,'end\n\n\n'];
end
```

```
%% Generating engine weight function
if k==1
    ResponseW = 'm_engine';
    FunctionName = ['Approx',ResponseW];
    FileName = [FunctionName,'.m'];
    FunctionFile=['function ',ResponseW,' = ',FunctionName,'(Afan,ro,ARm)\n'];
    FunctionFile=[FunctionFile,'%% This file was automatically generated.\n%% Model and
regression performed by Cyril de Tenorio\n%% Fall 2009 - Georgia Institute of
Technology\n','%% ',Time,'\n'];
    FunctionFile=[FunctionFile,'%%%% Inputs:\n%%-Afan: Fan face area [m2]\n%%-ro: Motor
rotor outer radius [mm]\n%%-ARm: machine aspect ratio [-]\n'];
    FunctionFile=[FunctionFile,'%%%%     Output:\n%%-',ResponseW,':     Engine     weight
[kg]\n\n\n'];
    % Print-in equation:
    FunctionFile = [FunctionFile,'r02 =',Net_bnd02,';\n',...
    'if ro<r02\n',...
    Offset,ResponseW,'= 0;\n',...
    'else\n',...
    Offset,ResponseW,' = ',Net_Weight,';\nend'];
    % Print to file
    fid = fopen(FileName,'wt');
    fprintf(fid,FunctionFile);
    fclose(fid);
end
end %Scenarios for loop

%% Creation of the ML functions files containing the surrogates
fid = fopen('ApproxThrustCap.m','wt');
fprintf(fid,ThrustCapFctFile);
fclose(fid);
fid = fopen('MotorPreSizing_reg.m','wt');
fprintf(fid,PreSizingMotor);
fclose(fid);
fid = fopen('ApproxEff.m','wt');
fprintf(fid,[EffFctInit,Eff_idleDef,EffMainFct,EffMaxSubFct,TRSubFct]);
fclose(fid);
```

Approxm_engine

```
function m_engine = Approxm_engine(Afan,ro,ARm)
% This file was automatically generated.
% Model and regression performed by Cyril de Tenorio
% Fall 2009 - Georgia Institute of Technology
% 31/10/2009 at 17:46:0
%% Inputs:
%-Afan: Fan face area [m2]
%-ro: Motor rotor outer radius [mm]
%-ARm: machine aspect ratio [-]
%% Output:
%-m_engine: Engine weight [kg]


r02   =157.1891507366553   +   172.3195443179458   *   1/(1+exp(-1*(-0.8369393143427   +
0.3779286961783 * Afan +   0.0088182178588 * ARm))) + 272.8776630233439 * 1/(1+exp(-1*(
0.8881702455986 + -0.2145083293144 * Afan + -0.1337825591247 * ARm))) + 19.3535003854515
* 1/(1+exp(-1*(-0.0400993135214 +  0.0038609836328 * Afan + -0.0226148354277 * ARm))) + -
381.5707936442569   *   1/(1+exp(-1*(-0.2618770519474   +    0.3349055267322   *   Afan   +   -
0.9003087740612   *   ARm)))   +   134.8087989032518   *   1/(1+exp(-1*(-0.0741371353300   +
0.0564292609766 * Afan + -0.0224756538791 * ARm))) + 681.2533343705154 * 1/(1+exp(-1*(
2.4007193515664   +    1.2029362215345   *   Afan   +    0.0429877002311   *   ARm)))   +   -
194.3781302253099   *   1/(1+exp(-1*(   1.3308570100311   +   -0.5650762528428   *   Afan   +
0.9968087372249   *   ARm)))   +   -240.2104477003967   *   1/(1+exp(-1*(   0.4936665980276   +   -
0.0486698718969 * Afan + -0.3031537023134 * ARm))) + 158.7250616481930 * 1/(1+exp(-1*(-
0.0381301781980   +    0.0656318419293   *   Afan   +   -0.0336226584546   *   ARm)))   +   -
802.6436255899782   *   1/(1+exp(-1*(   0.9270712770403   +   -0.1486335782614   *   Afan   +
0.6180592617343 * ARm)));
if ro<r02
    m_engine= 0;
else
    m_engine = -8501.3715118875352 + 3075.5409226539182 * 1/(1+exp(-1*(-4.1608421849607 +
-0.7141589717828 * Afan +   0.0241976586720 * ro +   0.2554314804635 * ARm))) + -
44314.8014405573410 * 1/(1+exp(-1*(  6.1818738511074 +  -0.0714046670310 * Afan + -
0.0058945768549 * ro + -0.2033379309991 * ARm))) + 42103.6252502312350 * 1/(1+exp(-1*(
1.8837346291384 +   0.2212224379394 * Afan + -0.0160847442759 * ro + -0.0986171560681 *
ARm))) + 12258.0527562209120 * 1/(1+exp(-1*( 0.8349472208067 +  0.1861828804793 * Afan +
-0.0034893018222 * ro +   0.2031132071529 * ARm))) + 8574.1471831414965 * 1/(1+exp(-1*(
0.2033003869290 + -0.4450064379364 * Afan +   0.0009558307379 * ro + -0.0306708760880 *
```

```
ARm))) + -25800.9931788064970 * 1/(1+exp(-1*(-0.6969611732509 + -0.0968938730914 * Afan +
-0.0014282727747 * ro +  0.1384980558589 * ARm))) + 15944.6651822319550 * 1/(1+exp(-1*(-
2.0436933469782 +  0.0648956144174 * Afan +  0.0023400113643 * ro +  0.2002435598651 *
ARm))) + 30846.1470471714570 * 1/(1+exp(-1*(-1.6918468273963 + -0.3377968742937 * Afan +
0.0186396891884 * ro +  0.1180579719095 * ARm))) + 9336.7818089206958 * 1/(1+exp(-1*(-
0.5131222474414 +  0.0066982274450 * Afan +  0.0046958407955 * ro + -0.0651448134125 *
ARm))) + 3083.0066649607415 * 1/(1+exp(-1*(-5.1672246146118 +  0.2367130451364 * Afan +
0.0271135514044 * ro +  0.0809771442914 * ARm)));
end
```

ApproxThrustCap

```
function ThCap=ApproxThrustCap(Afan,ro,ARm)
% This file was automatically generated.
% Model and regression performed by Cyril de Tenorio
% Fall 2009 - Georgia Institute of Technology
% 31/10/2009 at 17:46:0
%% Inputs:
%-Afan: Fan face area [m2]
%-ro: Motor rotor outer radius [mm]
%-ARm: machine aspect ratio [-]
%% Output:
%- ThCap: Max thrust capability at typical flight conditions


ThCap=zeros(1,8);

%% Representation of Max thrust capability at FL: 0 and MN: 0.04
r02 = [NET - removed for brievity];
if ro<r02
    ThCap(1)=0;
else
r12 = [NET - removed for brievity];
% Settings used for transition
    B_trans = 0.4;
    B_power = 5;
    rt = r12-B_trans*(r12-r02);% Transition radius
    if ro<rt
        Beta=0;
    elseif ro<r12
        Beta=((ro-rt)/(r12-rt))^B_power;
    else
        Beta=1;
    end
    ThCapZ1 = Afan *70500.344;
    if Beta<1
        ThCapZ2 = [NET - removed for brievity];
        if ThCapZ1 > ThCapZ2
            ThCap(1)= Beta*ThCapZ1 + (1-Beta)*ThCapZ2;
        else
            ThCap(1)=ThCapZ1;
        end
    else
        ThCap(1)=ThCapZ1;
    end
end


%% Representation of Max thrust capability at FL: 50 and MN: 0.41
r02 = [NET - removed for brievity];
if ro<r02
    ThCap(2)=0;
else
r12 = [NET - removed for brievity];
% Settings used for transition
    B_trans = 0.4;
    B_power = 5;
    rt = r12-B_trans*(r12-r02);% Transition radius
    if ro<rt
        Beta=0;
    elseif ro<r12
        Beta=((ro-rt)/(r12-rt))^B_power;
    else
        Beta=1;
    end
    ThCapZ1 = Afan *45977.948;
    if Beta<1
```

```
        ThCapZ2 = [NET - removed for brievity];
        if ThCapZ1 > ThCapZ2
            ThCap(2)= Beta*ThCapZ1 + (1-Beta)*ThCapZ2;
        else
            ThCap(2)=ThCapZ1;
        end
    else
        ThCap(2)=ThCapZ1;
    end
end


%% Representation of Max thrust capability at FL: 275 and MN: 0.75
r02 = [NET - removed for brievity];
if ro<r02
    ThCap(3)=0;
else
r12 = [NET - removed for brievity];
% Settings used for transition
    B_trans = 0.4;
    B_power = 5;
    rt = r12-B_trans*(r12-r02);% Transition radius
    if ro<rt
        Beta=0;
    elseif ro<r12
        Beta=((ro-rt)/(r12-rt))^B_power;
    else
        Beta=1;
    end
    ThCapZ1 = Afan *18161.341;
    if Beta<1
        ThCapZ2 = [NET - removed for brievity];
        if ThCapZ1 > ThCapZ2
            ThCap(3)= Beta*ThCapZ1 + (1-Beta)*ThCapZ2;
        else
            ThCap(3)=ThCapZ1;
        end
    else
        ThCap(3)=ThCapZ1;
    end
end


%% Representation of Max thrust capability at FL: 350 and MN: 0.78
r02 = [NET - removed for brievity];
if ro<r02
    ThCap(4)=0;
else
r12 = [NET - removed for brievity];
% Settings used for transition
    B_trans = 0.4;
    B_power = 5;
    rt = r12-B_trans*(r12-r02);% Transition radius
    if ro<rt
        Beta=0;
    elseif ro<r12
        Beta=((ro-rt)/(r12-rt))^B_power;
    else
        Beta=1;
    end
    ThCapZ1 = Afan *12950.755;
    if Beta<1
        ThCapZ2 = [NET - removed for brievity];
        if ThCapZ1 > ThCapZ2
            ThCap(4)= Beta*ThCapZ1 + (1-Beta)*ThCapZ2;
        else
            ThCap(4)=ThCapZ1;
        end
    else
        ThCap(4)=ThCapZ1;
    end
end


%% Representation of Max thrust capability at FL: 150 and MN: 0.71
r02 = [NET - removed for brievity];
if ro<r02
    ThCap(5)=0;
else
r12 = [NET - removed for brievity];
```

```
% Settings used for transition
    B_trans = 0.4;
    B_power = 5;
    rt = r12-B_trans*(r12-r02);% Transition radius
    if ro<rt
        Beta=0;
    elseif ro<r12
        Beta=((ro-rt)/(r12-rt))^B_power;
    else
        Beta=1;
    end
    ThCapZ1 = Afan *30553.767;
    if Beta<1
        ThCapZ2 = [NET - removed for brievity];
        if ThCapZ1 > ThCapZ2
            ThCap(5)= Beta*ThCapZ1 + (1-Beta)*ThCapZ2;
        else
            ThCap(5)=ThCapZ1;
        end
    else
        ThCap(5)=ThCapZ1;
    end
end


%% Representation of Max thrust capability at FL: 100 and MN: 0.6
r02 = [NET - removed for brievity];if ro<r02
    ThCap(6)=0;
else
r12 = [NET - removed for brievity];
% Settings used for transition
    B_trans = 0.4;
    B_power = 5;
    rt = r12-B_trans*(r12-r02);% Transition radius
    if ro<rt
        Beta=0;
    elseif ro<r12
        Beta=((ro-rt)/(r12-rt))^B_power;
    else
        Beta=1;
    end
    ThCapZ1 = Afan *36839.311;
    if Beta<1
        ThCapZ2 = [NET - removed for brievity];
        if ThCapZ1 > ThCapZ2
            ThCap(6)= Beta*ThCapZ1 + (1-Beta)*ThCapZ2;
        else
            ThCap(6)=ThCapZ1;
        end
    else
        ThCap(6)=ThCapZ1;
    end
end


%% Representation of Max thrust capability at FL: 100 and MN: 0.45
r02 = [NET - removed for brievity];    ThCap(7)=0;
else
r12 = [NET - removed for brievity];
% Settings used for transition
    B_trans = 0.4;
    B_power = 5;
    rt = r12-B_trans*(r12-r02);% Transition radius
    if ro<rt
        Beta=0;
    elseif ro<r12
        Beta=((ro-rt)/(r12-rt))^B_power;
    else
        Beta=1;
    end
    ThCapZ1 = Afan *37601.437;
    if Beta<1
        ThCapZ2 = [NET - removed for brievity];
        if ThCapZ1 > ThCapZ2
            ThCap(7)= Beta*ThCapZ1 + (1-Beta)*ThCapZ2;
        else
            ThCap(7)=ThCapZ1;
        end
    else
        ThCap(7)=ThCapZ1;
```

```
    end
end


%% Representation of Max thrust capability at FL: 0 and MN: 0.3
r02 =[NET - removed for brievity]if ro<r02
    ThCap(8)=0;
else
r12 = [NET - removed for brievity]
% Settings used for transition
    B_trans = 0.4;
    B_power = 5;
    rt = r12-B_trans*(r12-r02);% Transition radius
    if ro<rt
        Beta=0;
    elseif ro<r12
        Beta=((ro-rt)/(r12-rt))^B_power;
    else
        Beta=1;
    end
    ThCapZ1 = Afan *57713.597;
    if Beta<1
        ThCapZ2 = [NET - removed for brievity];
        if ThCapZ1 > ThCapZ2
            ThCap(8)= Beta*ThCapZ1 + (1-Beta)*ThCapZ2;
        else
            ThCap(8)=ThCapZ1;
        end
    else
        ThCap(8)=ThCapZ1;
    end
end
```

ApproxEff

```
function [Eff,EffMax,TR] = ApproxEff(Afan,ro,ARm,Treq,Tcap,Cond)
% This file was automatically generated.
% Model and regression performed by Cyril de Tenorio
% Fall 2009 - Georgia Institute of Technology
% 31/10/2009 at 19:55:23
%% Inputs:
%-Afan: Fan face area [m2]
%-ro: Motor rotor outer radius [mm]
%-ARm: machine aspect ratio [-]
%-Treq: Thrust required [N]
%-Tcap: Max Thrust capacity in the given flight condition [N]
%% Output:
%-Eff: Efficiency of the electric ducted fan providing Treq N of thrust at Flight
Conditions "Cond".
% The flight conditions are defined as follows:
% Cond         Altitude        MachNumber
% 1             0             0.04% 2             5000             0.41% 3             27500
0.75% 4           35000           0.78% 5           15000           0.71% 6           10000
0.6% 7           10000          0.45% 8          0          0.3

Eff_idleTable = [0.16433;0.69713;0.77776;0.78153;0.77219;0.75294;0.71272;0.63481];

TR = ApproxTR(Afan,ro,ARm,Cond);   % ratio idle/max thrust
Eff_idle = Eff_idleTable(Cond);   % efficiency of fan at idle
TRreq=Treq/Tcap;                % thrust level at which the fan is operating
if TRreq<TR % The fan is operating under idle speed
    Eff = Eff_idle;
    EffMax=0;
else
    EffMax=ApproxEffMax(Afan,ro,ARm,Cond);
    Eff = Eff_idle+(EffMax-Eff_idle)/(1-TR)*(TRreq-TR);
end


function EffMax = ApproxEffMax(Afan,ro,ARm,Cond)
%Output:EffMaxOverall efficiency at max thrust regime [-]

if Cond==1
%% Representation of Max thrust efficiency at FL: 0 and MN: 0.04
    r02 = '[NET - removed for brievity]';
    if ro<r02
        EffMax = 0;
    else
```

```
         r12 = '[NET - removed for brievity]'    ;
        % Settings used for transition
        B_trans = 0.4;
        B_power = 5;
        rt = r12-B_trans*(r12-r02);% Transition radius
        if ro<rt
            Beta=0;
        elseif ro<r12
            Beta=((ro-rt)/(r12-rt))^B_power;
        else
            Beta=1;
        end
        EffMax1Z1 = 0.067098;
        if Beta<1    % The design is in zone 2
            EffMax1Z2 = '[NET - removed for brievity]';
            if EffMax1Z2 > EffMax1Z1
                EffMax = Beta*EffMax1Z1 + (1-Beta)*EffMax1Z2;
            else    % The design is in zone 1
                EffMax=EffMax1Z1;
            end
        else
            EffMax=EffMax1Z1;
        end
    end
elseif Cond==2
%% Representation of Max thrust efficiency at FL: 50 and MN: 0.41
    r02 = '[NET - removed for brievity]';
    if ro<r02
        EffMax = 0;
    else
         r12 = '[NET - removed for brievity]'    ;
        % Settings used for transition
        B_trans = 0.4;
        B_power = 5;
        rt = r12-B_trans*(r12-r02);% Transition radius
        if ro<rt
            Beta=0;
        elseif ro<r12
            Beta=((ro-rt)/(r12-rt))^B_power;
        else
            Beta=1;
        end
        EffMax2Z1 = 0.47205;
        if Beta<1    % The design is in zone 2
            EffMax2Z2 = '[NET - removed for brievity]';
            if EffMax2Z2 > EffMax2Z1
                EffMax = Beta*EffMax2Z1 + (1-Beta)*EffMax2Z2;
            else    % The design is in zone 1
                EffMax=EffMax2Z1;
            end
        else
            EffMax=EffMax2Z1;
        end
    end
elseif Cond==3
%% Representation of Max thrust efficiency at FL: 275 and MN: 0.75
    r02 = '[NET - removed for brievity]';
    if ro<r02
        EffMax = 0;
    else
         r12 = '[NET - removed for brievity]'    ;
        % Settings used for transition
        B_trans = 0.4;
        B_power = 5;
        rt = r12-B_trans*(r12-r02);% Transition radius
        if ro<rt
            Beta=0;
        elseif ro<r12
            Beta=((ro-rt)/(r12-rt))^B_power;
        else
            Beta=1;
        end
        EffMax3Z1 = 0.63429;
        if Beta<1    % The design is in zone 2
            EffMax3Z2 = '[NET - removed for brievity]';
            if EffMax3Z2 > EffMax3Z1
                EffMax = Beta*EffMax3Z1 + (1-Beta)*EffMax3Z2;
            else    % The design is in zone 1
                EffMax=EffMax3Z1;
            end
```

```
        else
            EffMax=EffMax3Z1;
        end
    end
elseif Cond==4
%% Representation of Max thrust efficiency at FL: 350 and MN: 0.78
    rO2 = '[NET - removed for brievity]';
    if ro<rO2
        EffMax = 0;
    else
         r12 = '[NET - removed for brievity]'    ;
        % Settings used for transition
        B_trans = 0.4;
        B_power = 5;
        rt = r12-B_trans*(r12-rO2);% Transition radius
        if ro<rt
            Beta=0;
        elseif ro<r12
            Beta=((ro-rt)/(r12-rt))^B_power;
        else
            Beta=1;
        end
        EffMax4Z1 = 0.64267;
        if Beta<1    % The design is in zone 2
            EffMax4Z2 = '[NET - removed for brievity]';
            if EffMax4Z2 > EffMax4Z1
                EffMax = Beta*EffMax4Z1 + (1-Beta)*EffMax4Z2;
            else    % The design is in zone 1
                EffMax=EffMax4Z1;
            end
        else
            EffMax=EffMax4Z1;
        end
    end
elseif Cond==5
%% Representation of Max thrust efficiency at FL: 150 and MN: 0.71
    rO2 = '[NET - removed for brievity]';
    if ro<rO2
        EffMax = 0;
    else
         r12 = '[NET - removed for brievity]'    ;
        % Settings used for transition
        B_trans = 0.4;
        B_power = 5;
        rt = r12-B_trans*(r12-rO2);% Transition radius
        if ro<rt
            Beta=0;
        elseif ro<r12
            Beta=((ro-rt)/(r12-rt))^B_power;
        else
            Beta=1;
        end
        EffMax5Z1 = 0.62174;
        if Beta<1    % The design is in zone 2
            EffMax5Z2 = '[NET - removed for brievity]';
            if EffMax5Z2 > EffMax5Z1
                EffMax = Beta*EffMax5Z1 + (1-Beta)*EffMax5Z2;
            else    % The design is in zone 1
                EffMax=EffMax5Z1;
            end
        else
            EffMax=EffMax5Z1;
        end
    end
elseif Cond==6
%% Representation of Max thrust efficiency at FL: 100 and MN: 0.6
    rO2 = '[NET - removed for brievity]';
    if ro<rO2
        EffMax = 0;
    else
         r12 = '[NET - removed for brievity]'    ;
        % Settings used for transition
        B_trans = 0.4;
        B_power = 5;
        rt = r12-B_trans*(r12-rO2);% Transition radius
        if ro<rt
            Beta=0;
        elseif ro<r12
            Beta=((ro-rt)/(r12-rt))^B_power;
        else
```

```
            Beta=1;
        end
        EffMax6Z1 = 0.57888;
        if Beta<1    % The design is in zone 2
            EffMax6Z2 = '[NET - removed for brievity]';
            if EffMax6Z2 > EffMax6Z1
                EffMax = Beta*EffMax6Z1 + (1-Beta)*EffMax6Z2;
            else    % The design is in zone 1
                EffMax=EffMax6Z1;
            end
        else
            EffMax=EffMax6Z1;
        end
    end
elseif Cond==7
%% Representation of Max thrust efficiency at FL: 100 and MN: 0.45
    r02 = '[NET - removed for brievity]';
    if ro<r02
        EffMax = 0;
    else
         r12 = '[NET - removed for brievity]'    ;
        % Settings used for transition
        B_trans = 0.4;
        B_power = 5;
        rt = r12-B_trans*(r12-r02);% Transition radius
        if ro<rt
            Beta=0;
        elseif ro<r12
            Beta=((ro-rt)/(r12-rt))^B_power;
        else
            Beta=1;
        end
        EffMax7Z1 = 0.49861;
        if Beta<1    % The design is in zone 2
            EffMax7Z2 = '[NET - removed for brievity]';
            if EffMax7Z2 > EffMax7Z1
                EffMax = Beta*EffMax7Z1 + (1-Beta)*EffMax7Z2;
            else    % The design is in zone 1
                EffMax=EffMax7Z1;
            end
        else
            EffMax=EffMax7Z1;
        end
    end
elseif Cond==8
%% Representation of Max thrust efficiency at FL: 0 and MN: 0.3
    r02 = '[NET - removed for brievity]';
    if ro<r02
        EffMax = 0;
    else
         r12 = '[NET - removed for brievity]'    ;
        % Settings used for transition
        B_trans = 0.4;
        B_power = 5;
        rt = r12-B_trans*(r12-r02);% Transition radius
        if ro<rt
            Beta=0;
        elseif ro<r12
            Beta=((ro-rt)/(r12-rt))^B_power;
        else
            Beta=1;
        end
        EffMax8Z1 = 0.38503;
        if Beta<1    % The design is in zone 2
            EffMax8Z2 = '[NET - removed for brievity]';
            if EffMax8Z2 > EffMax8Z1
                EffMax = Beta*EffMax8Z1 + (1-Beta)*EffMax8Z2;
            else    % The design is in zone 1
                EffMax=EffMax8Z1;
            end
        else
            EffMax=EffMax8Z1;
        end
    end
end


function TR = ApproxTR(Afan,ro,ARm,Cond)
%Output:TRIdle to max thrust ratio [-]
if Cond==1
```

```
%% Representation of Max thrust efficiency at FL: 0 and MN: 0.04
    r02 = '[NET - removed for brievity]';
    if ro<r02
        TR = 0;
    else
         r12 = '[NET - removed for brievity]'    ;
        % Settings used for transition
        B_trans = 0.4;
        B_power = 5;
        rt = r12-B_trans*(r12-r02);% Transition radius
        if ro<rt
            Beta=0;
        elseif ro<r12
            Beta=((ro-rt)/(r12-rt))^B_power;
        else
            Beta=1;
        end
        TR1Z1 = 0.11097;
        if Beta<1    % The design is in zone 2
            TR1Z2 = '[NET - removed for brievity]';
            if TR1Z2 > TR1Z1
                TR = Beta*TR1Z1 + (1-Beta)*TR1Z2;
            else    % The design is in zone 1
                TR=TR1Z1;
            end
        else
            TR=TR1Z1;
        end
    end

elseif Cond==2
%% Representation of Max thrust efficiency at FL: 50 and MN: 0.41
    r02 = '[NET - removed for brievity]';
    if ro<r02
        TR = 0;
    else
         r12 = '[NET - removed for brievity]'    ;
        % Settings used for transition
        B_trans = 0.4;
        B_power = 5;
        rt = r12-B_trans*(r12-r02);% Transition radius
        if ro<rt
            Beta=0;
        elseif ro<r12
            Beta=((ro-rt)/(r12-rt))^B_power;
        else
            Beta=1;
        end
        TR2Z1 = 0.066909;
        if Beta<1    % The design is in zone 2
            TR2Z2 = '[NET - removed for brievity]';
            if TR2Z2 > TR2Z1
                TR = Beta*TR2Z1 + (1-Beta)*TR2Z2;
            else    % The design is in zone 1
                TR=TR2Z1;
            end
        else
            TR=TR2Z1;
        end
    end

elseif Cond==3
%% Representation of Max thrust efficiency at FL: 275 and MN: 0.75
    r02 = '[NET - removed for brievity]';
    if ro<r02
        TR = 0;
    else
         r12 = '[NET - removed for brievity]'    ;
        % Settings used for transition
        B_trans = 0.4;
        B_power = 5;
        rt = r12-B_trans*(r12-r02);% Transition radius
        if ro<rt
            Beta=0;
        elseif ro<r12
            Beta=((ro-rt)/(r12-rt))^B_power;
        else
            Beta=1;
        end
        TR3Z1 = 0.05555;
```

```
        if Beta<1    % The design is in zone 2
            TR3Z2 = '[NET - removed for brievity]';
            if TR3Z2 > TR3Z1
                TR = Beta*TR3Z1 + (1-Beta)*TR3Z2;
            else    % The design is in zone 1
                TR=TR3Z1;
            end
        else
            TR=TR3Z1;
        end
    end

elseif Cond==4
%% Representation of Max thrust efficiency at FL: 350 and MN: 0.78
    rO2 = '[NET - removed for brievity]';
    if ro<rO2
        TR = 0;
    else
         r12 = '[NET - removed for brievity]'    ;
        % Settings used for transition
        B_trans = 0.4;
        B_power = 5;
        rt = r12-B_trans*(r12-rO2);% Transition radius
        if ro<rt
            Beta=0;
        elseif ro<r12
            Beta=((ro-rt)/(r12-rt))^B_power;
        else
            Beta=1;
        end
        TR4Z1 = 0.055089;
        if Beta<1    % The design is in zone 2
            TR4Z2 = '[NET - removed for brievity]';
            if TR4Z2 > TR4Z1
                TR = Beta*TR4Z1 + (1-Beta)*TR4Z2;
            else    % The design is in zone 1
                TR=TR4Z1;
            end
        else
            TR=TR4Z1;
        end
    end

elseif Cond==5
%% Representation of Max thrust efficiency at FL: 150 and MN: 0.71
    rO2 = '[NET - removed for brievity]';
    if ro<rO2
        TR = 0;
    else
         r12 = '[NET - removed for brievity]'    ;
        % Settings used for transition
        B_trans = 0.4;
        B_power = 5;
        rt = r12-B_trans*(r12-rO2);% Transition radius
        if ro<rt
            Beta=0;
        elseif ro<r12
            Beta=((ro-rt)/(r12-rt))^B_power;
        else
            Beta=1;
        end
        TR5Z1 = 0.056267;
        if Beta<1    % The design is in zone 2
            TR5Z2 = '[NET - removed for brievity]';
            if TR5Z2 > TR5Z1
                TR = Beta*TR5Z1 + (1-Beta)*TR5Z2;
            else    % The design is in zone 1
                TR=TR5Z1;
            end
        else
            TR=TR5Z1;
        end
    end

elseif Cond==6
%% Representation of Max thrust efficiency at FL: 100 and MN: 0.6
    rO2 = '[NET - removed for brievity]';
    if ro<rO2
        TR = 0;
    else
```

```
         r12 = '[NET - removed for brievity]'    ;
        % Settings used for transition
        B_trans = 0.4;
        B_power = 5;
        rt = r12-B_trans*(r12-r02);% Transition radius
        if ro<rt
            Beta=0;
        elseif ro<r12
            Beta=((ro-rt)/(r12-rt))^B_power;
        else
            Beta=1;
        end
        TR6Z1 = 0.058933;
        if Beta<1    % The design is in zone 2
            TR6Z2 = '[NET - removed for brievity]';
            if TR6Z2 > TR6Z1
                TR = Beta*TR6Z1 + (1-Beta)*TR6Z2;
            else    % The design is in zone 1
                TR=TR6Z1;
            end
        else
            TR=TR6Z1;
        end
    end

elseif Cond==7
%% Representation of Max thrust efficiency at FL: 100 and MN: 0.45
    r02 = '[NET - removed for brievity]';
    if ro<r02
        TR = 0;
    else
         r12 = '[NET - removed for brievity]'    ;
        % Settings used for transition
        B_trans = 0.4;
        B_power = 5;
        rt = r12-B_trans*(r12-r02);% Transition radius
        if ro<rt
            Beta=0;
        elseif ro<r12
            Beta=((ro-rt)/(r12-rt))^B_power;
        else
            Beta=1;
        end
        TR7Z1 = 0.064759;
        if Beta<1    % The design is in zone 2
            TR7Z2 = '[NET - removed for brievity]';
            if TR7Z2 > TR7Z1
                TR = Beta*TR7Z1 + (1-Beta)*TR7Z2;
            else    % The design is in zone 1
                TR=TR7Z1;
            end
        else
            TR=TR7Z1;
        end
    end

elseif Cond==8
%% Representation of Max thrust efficiency at FL: 0 and MN: 0.3
    r02 = '[NET - removed for brievity]';
    if ro<r02
        TR = 0;
    else
         r12 = '[NET - removed for brievity]'    ;
        % Settings used for transition
        B_trans = 0.4;
        B_power = 5;
        rt = r12-B_trans*(r12-r02);% Transition radius
        if ro<rt
            Beta=0;
        elseif ro<r12
            Beta=((ro-rt)/(r12-rt))^B_power;
        else
            Beta=1;
        end
        TR8Z1 = 0.074705;
        if Beta<1    % The design is in zone 2
            TR8Z2 = '[NET - removed for brievity]';
            if TR8Z2 > TR8Z1
                TR = Beta*TR8Z1 + (1-Beta)*TR8Z2;
            else    % The design is in zone 1
```

```
                TR=TR8Z1;
            end
        else
            TR=TR8Z1;
        end
    end
end
```

CompareApproxToReal

```
clc
clear all
close all

%% Mission points
np = 25;

%% Design variables Ranges
% DV(1) = Afan    % DV(2) = ro    % DV(3) = ARm
r=1;% A_fan - Fan Area [m2] - Typical range [0.2-9]
lb(r)= 0.2;      ub(r)= 9;   npt(r)=np;
r=2;% ro  - Mean armature radius  (mm) - Typical range [65-300] Nom 140
lb(r)= 65;       ub(r)= 320;   npt(r)=np;
r=3;% ARm - Machine shape factor (La/ro) - Typical range [1.6-8]
lb(r)= 1.6;        ub(r)= 8;   npt(r)=np;
nDV=r;           % Total number of design variables

%% Build full factorial Design of Experiement
Xlevels = zeros(nDV,max(npt));
for dv=1:nDV
    Xlevels(dv,:)=linspace(lb(dv),ub(dv),npt(dv));
end

X=zeros(3,prod(npt));
Nexp=0;%Number of experiements in the DOE
for dv1=1:npt(1)
    for dv2=1:npt(2)
        for dv3=1:npt(3)
            Nexp=Nexp+1;
            X(:,Nexp)=[Xlevels(1,dv1);Xlevels(2,dv2);Xlevels(3,dv3)];
        end
    end
end
% Preparation of response matrices
Table=zeros(Nexp,3+3+3+1);

%% Dimensions to be observed
ifARm=Xlevels(3,20);
Cond=1;

for n=1:Nexp
    %% Get responses
    ThCap = ApproxThrustCap(X(1,n),X(2,n),X(3,n));
    m_engine = Approxm_engine(X(1,n),X(2,n),X(3,n));
    [Eff3,EffMax,TR] = ApproxEff(X(1,n),X(2,n),X(3,n),1,1,Cond);
    Table(n,1:7)=[X(1,n),X(2,n),X(3,n),ThCap(Cond),EffMax,TR,m_engine];
end
[Data]=GetDoEData();
K=8;

% Importing training data
k=Cond;
rd(1)=8;         Table(:,rd(1))=Data(:,k);  %Thrust
rd(2)=rd(1)+1;   Table(:,rd(2))=Data(:,k+K+1); %EffMax
rd(3)=rd(2)+1;   Table(:,rd(3))=Data(:,k+3*K+1); %TR
rd(4)=rd(3)+1;   Table(:,rd(4))=Data(:,K+1); %m_engine

%% Plot
r=0;
figure(1)
subplot(2,2,1);view(-45,45);title('Thrust');
subplot(2,2,2);view(-45,45);title('Effmax');
subplot(2,2,3);view(-45,45);title('TR');
subplot(2,2,4);view(-45,45);title('Engine mass')
for n=1:Nexp
    if Table(n,3)==ifARm
        for i=1:4
            hold on;
```

```
            x1=Table(n,1);x2=Table(n,2);
            z =Table(n,rd(i));zapp=Table(n,3+i);
            subplot(2,2,i);
            plot3(x1,x2,zapp,'b.')
            if zapp>z
                plot3([x1,x1],[x2,x2],[z,zapp],'-r')
            else
                plot3([x1,x1],[x2,x2],[z,zapp],'-g')
            end
        end
    end
end
figure(2)
view(-45,45)
subplot(2,2,1);view(-45,45);%title(['Thrust Capacity at ARm=',num2str(ifARm)])
subplot(2,2,2);view(-45,45);%title(['Eff at max thrust at ARm=',num2str(ifARm)]')
subplot(2,2,3);view(-45,45);%title(['Idle thrust ratio at ARm=',num2str(ifARm)]')
subplot(2,2,4);view(-45,45);%title(['Engine mass at ARm =',num2str(ifARm)]')
for n=1:Nexp
    if Table(n,3)==ifARm
        for i=1:4
            x1=Table(n,1);x2=Table(n,2);
            z =Table(n,rd(i));zapp=Table(n,3+i);
            hold on;
            subplot(2,2,i);
            plot3(x1,x2,z,'g.')
        end
    end
end
```

Limitscanner.m

```
% Grows the functional requirement until either AfanMax or roMax are
% reached by the presizing models.
% This routine is used to determine the range of application of the
% electric fan sizing model.

clc
clear all
close all

AfanMax = 9; %[m2] Max fan area allowable
roMax = 320; % [mm]Max motor radius allowable

ItMax = 100;
Maxthrust_cond = [38375,26625,8812.50000000000,8450,8975,10525,9775,10000];

K=length(Maxthrust_cond);
ARmDefault = 4.; %Motor aspect ratio used in the pre-sizing
ScannInc = 1.05; %Each step will increase the tested value geometrically by multiplying
by this term

ValidThMax=zeros(K,1);
ThScan=zeros(K,ItMax);AfanTable=zeros(K,ItMax);roStartTable=zeros(K,ItMax);
for k=1:K
    ThScan(k,1)=Maxthrust_cond(k); %initialize scanning process

    it=1;Validity=1;
    while it<ItMax &&Validity==1
        it=it+1;
        ThScan(k,it)=ThScan(k,1)*ScannInc^it;
        Maxthrust_cond(k)=ThScan(k,it);
        Offs = 1.1;% Offset used to oversize the initial guess
        Afanmin = FanPreSizing_reg(Maxthrust_cond*Offs,0);
        if Afanmin>AfanMax
            Validity=0;
            ValidThMax(k)=ThScan(k,it-1);
        else
            [roStart] = MotorPreSizing_reg(Afanmin,ARmDefault,0);
            if roStart>roMax
                Validity=0;
                ValidThMax(k)=ThScan(k,it-1);
            else
                AfanTable(k,it)=Afanmin;
                roStartTable(k,it)=roStart;
            end

        end
        if it==ItMax
            disp(['No lim found for k=',num2str(k),'\n']);
        end
    end
    Maxthrust_cond(k)=ThScan(k,1);
end
```

Appendix C

Sizing the Power Plant

C.1 General Description of the System

The power plant is the subsystem which transforms the chemical energy stored in the fuel into "useable" mechanical energy. The type of power plan considered in this thesis is a gas turbine. The gas turbine is the association of a compressor unit, a combustor and a turbine unit. This device is assumed to operate on a Bayton cycle. The fuel is injected in the flow in the combustor. Power is extracted using a turbine connected to a gear-box.

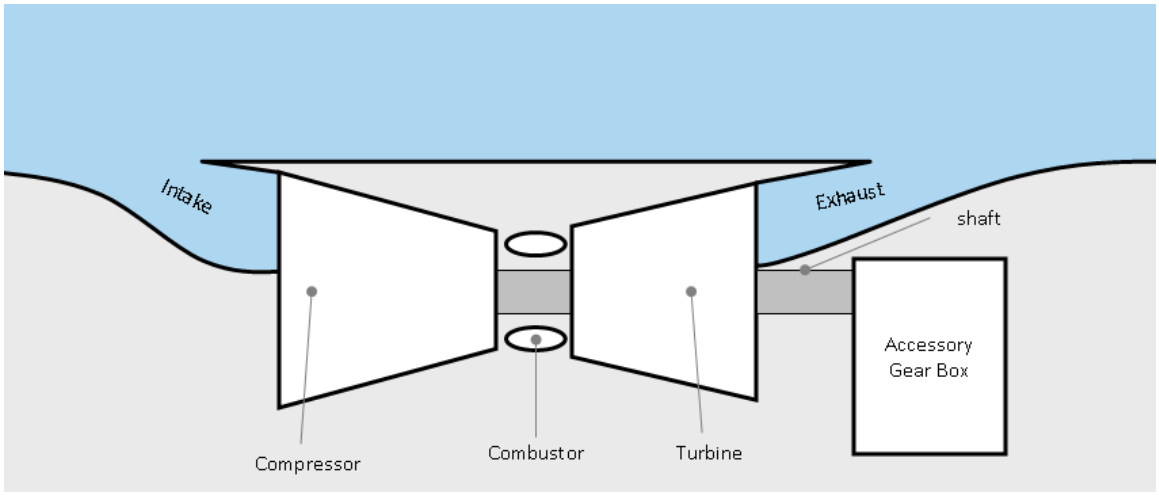


Figure 286: Description of the power plant subsystem

The analysis of the power plant described in this appendix is merely a place holder and is not based on experimental observations or historical data. The analysis provided to size and represent the power plant is only based on rudimentary Brayton cycle relationships.

C.2 Description of the Sizing Model

The power plant sizing model is an optimizer-based sizing model constructed around the following optimization problem:

$$\underset{\overline{X}}{Min} \quad \gamma_1 \frac{W_{PP}\left(\overline{X}\right)}{W_{PP-baseline}} + \gamma_2 \frac{FuelFlow_{@\,cruise}\left(\overline{X}\right)}{FuelFlow_{@\,cruise-baseline}} \tag{123}$$

Subject to: $Cap_{@k} > LoadM_{@k}$ for all mission state k

In this equation $Cap_{@k}$ refers to the maximum power off-take that can be accommodated by the power plant in flight condition k and $LoadM_{@k}$ is effective peak power requirement in mission scenario k.

Two design variables were used in this sizing problem. The first was the air mass flow rate of the engine; the second was the overall pressure ratio π of the compressor. In order to match the power off-take the mass flow rate was directly determined from the peak load and π using equation (124).

$$W_a = \frac{LoadM\big|_{\max}}{Cp\left[T4\times\left(1-\pi^{-\frac{\gamma-1}{\gamma}}\right)-T2\times\pi^{\frac{\gamma-1}{\gamma}}-1\right]} \tag{124}$$

In equation (124), Cp refers the heat capacity of air, T4 the turbine inlet temperature (technology factor) and T2 is the ambient air. The efficiency and fuel mass flow rate of the engine is defined by the follows:

$$\eta = 1-\pi^{-\frac{\gamma-1}{\gamma}} \tag{125}$$

$$mf_k = \begin{cases} Max\left(\dfrac{Loadnom_k}{\eta\times Qf}, 0.05\times\dfrac{LoadM\big|_{\max}}{\eta\times Qf}\right) & \text{if } LoadM_k > 0 \\ 0 & \text{if } LoadM_k = 0 \end{cases} \tag{126}$$

Qf refers to the heat capacity of the fuel. The conditional definition of the mass flow of fuel is used to represent the fuel burn associated with idling engine. This idle fuel flow is assumed to be 5% of the max fuel burn.

The weight estimation is decomposed on three items:

- The accessory gearbox weight (W_{AGB})
- The nacelle weight ($W_{nacelle}$)
- The compressor/turbine weight (W_{CT})

These weight estimations are based on the following equations:

$$W_{AGB} = 1\left[{kg}/{kW}\right] \times LoadM\big|_{\max} \quad \textbf{(127)}$$

$$W_{nacelle} = 15\left[{kg}/{(kg/s)}\right] \times W_a \quad \textbf{(128)}$$

$$W_{CT} = 3.5\left[{kg}/{(kg/s)}\right] \times W_a \times \pi \quad \textbf{(129)}$$

C.3 Code

The code used to represent the engine is presented in the following box:

```
function [W,mf,Wa]=GasTurbineF(OPR,DT,Cruisek,T41,LoadM,Loadm,RPMnom,SpRat,Ta,Pa,Ma)

%% INPUTS
% Tech k-factors
% T41     :       Maximum turbine inlet temperature [K]
% DT      :       Duration of each time step [min]
% OPR     :       Overall pressure ratio
%Mechanical variable interfaces
% LoadM   :       Maximum shaft power to be produced [kW]
% Loadm   :       Nominal shaft power to be produced [kW]
% RPMnom  :       Shaft speed delivered [Rot./min]
% SpRat   :       Speed ratio (f_max/f_min)
%
%Ambiant variables
% Ta      :       Ambient temperature [oC]
% Pa      :       Ambient pressure [Pa]
% Ma      :       Flight Mach number [-]
%% OUTPUTS
% Attributes
% Weight  :       Gas Turbine weight [kg] (scalar attribute)
% mf      :       Fuel flow [kg/s](vector attribute)
%%  Model
% Technology factors
T4=T41;                         % Maximum turbine inlet temperature [K]
Qf=43.15*10^3;                  % Fuel specific energy [kJ/kg]
SpeW_AGB = 1;                   % Accessory Gearbox specific weight [kg/kW]
SpeW_nacelle = 15;              % Nacelle specific weight [kg/(kg/s) air flow]
SpeW_comp = 3.5;                % Compressor weight[kg/airflow/OPR]


gamma= 1.4;                     % Gas constant
Cp= 1.005;                      % Specific heat of Air [kJ/kg/K]
```

```
Chi = (gamma-1)/gamma;          % Isentropic factor

PowMax=max(LoadM);              % Max load rating of the turbine

T2=Ta(1);
Wa = PowMax/(Cp*(T4*(1-OPR^(-Chi))-T2*(OPR^Chi-1)));

W = SpeW_AGB*PowMax+...
        Wa*SpeW_nacelle+...
        Wa*OPR*SpeW_comp;
K=length(LoadM);
mf=zeros(1,K);
eff=1-OPR^(-Chi);
mfIdle=PowMax/eff/Qf*0.05;
for k=1:K
    if LoadM(k)>0 && DT(k)>0
        mf(k) = max(Loadm(k)/eff/Qf,mfIdle);
    end
end
```

In order to perform the optimizer-based sizing of the power plant, the analysis above was adapted in the following function:

```
function [OEC,W,mfc,Wa]=GasTurbineDM(OPR)

global DT Cruisek LoadM Loadnom T41 RPMnom SpRat Ta Pa Ma W_b mf_b gammas

[W,mf,Wa]=GasTurbineF(OPR,DT,Cruisek,T41,LoadM,Loadnom,RPMnom,SpRat,Ta,Pa,Ma);

mfc=mf(Cruisek);
OEC = gammas(1)*W/W_b+gammas(2)*mfc/mf_b;
```

The interface function including the Matlab optimizer *fminbnd* is presented in the following box. It is important to observe that the overall pressure ratio π was bounded between a ratio of 10 and 30. Also the baseline values used in the objective functions correspond to the attributes of an engine operating with a pressure ratio of 15.

```
function
[W,FB]=GasTurbine(DT_,Cruisek_,show,T41_,LoadM_,Loadnom_,RPMnom_,SpRat_,Ta_,Pa_,Ma_,gamma
s_)
global DT Cruisek T41 LoadM Loadnom RPMnom SpRat Ta Pa Ma W_b mf_b gammas

Testingmode=0;      % should be 0 unless you are testing the function
%% for testing Make sure you comment the function line
if Testingmode~=0
    DT=[1,2];
    Cruisek=1;
    T41=1800;
    LoadM=[22,10]*1000;
    Loadnom=[20,8]*1000;
    RPMnom=[1000,1000];
    SpRat=[1.8,1.8];
    Ta=290;
    Pa=[1000000,1000000];
    Ma=[0.2,0.3];
    gammas(1)=0.4;
    gammas(2)=1-gammas(1);
    Pplot=1;% set this to 1 if you want to observe trade-off
else
    DT=DT_;
    Cruisek=Cruisek_;
    T41=T41_;
    LoadM=LoadM_;
    Loadnom=Loadnom_;
    RPMnom=RPMnom_;
    SpRat=SpRat_;
    Ta=Ta_;
    Pa=Pa_;
    Ma=Ma_;
    gammas=gammas_;
end
```

```
%% Definition of baseline engine
BaselineOPR= 15;
[W_b,mf]=GasTurbineF(BaselineOPR,DT,Cruisek,T41,LoadM,Loadnom,RPMnom,SpRat,Ta,Pa,Ma);
mf_b=mf(Cruisek);

%% Optimization based on OEC
OPR_best=fminbnd(@GasTurbineDM,10,30);

%% Return the attributes of the best engine
[W,mf]=GasTurbineF(OPR_best,DT,Cruisek,T41,LoadM,Loadnom,RPMnom,SpRat,Ta,Pa,Ma);
FB=zeros(1,length(DT));
for k=1:length(DT)
    FB(k)=mf(k)*DT(k);
end
```

Appendix D

Sizing the Electric Generators

D.1 General Description of the System

The electric generators are devices that transform mechanical energy into electric. In the test-case two types of generators are considered. The first type is the Direct Current electric generators (DCG). This type is used primarily for propulsive power. Therefore this generator will be assumed to have a high specific power with a slightly lower efficiency compared to the other type. The other type corresponds to the Variable Frequency generators (VFG). This type of generator is only used for non-propulsive purposes and is assumed to be a little more power extensive that the DCG.

D.2 Description of the Sizing Model

Both generators types are sized based on a similar functional regression approach presented in the following equations:

$$req_k = \frac{Load_k}{\eta} \tag{130}$$

$$W = \frac{Load\big|_{\max}}{PowerDensity} \tag{131}$$

These basic equations (where req quantifies the mechanical power required by the electric generator and Load the electric power load) define the power required by the electric generators and the estimate the weight of the devices. The parameter used for the power density and the efficiencies are indicated in the following table:

	η	Power density
DCG	0.90	8.73 kg/kW
VFG	0.95	3.6 kg/kW

D.3 Code

The code associated with the sizing of each of these subsystem types are indicated in the following boxes. The first box shows the code associated with the DCG:

```
function [Mass,reqM,reqm,HRM,HRm]=DCG(DT,Cruisek,show,Vdel,LoadM,Loadm,RPMnom,SpRat)

%% Model itself
% Technology factors
PowDen = 8.73;          % Power density of the genator [kW/kg]
keff = 0.90;             % Generator efficiency

% Technological constraints
K = length(Vdel);% # of scenarios
Pmax = max(LoadM); % Power capacity of the generator [kW]

error =0;% change to 1 if a requirement is infeasible
A=zeros(K,1);
reqM=A;reqm=A;HRM=A;HRm=A;

for k=1:K
    eff = keff; % This could be expressed as a function of speed and load

    if LoadM(k)>0 && RPMnom(k)==0
        error=1;
    end
    reqM(k)= LoadM(k)/eff; % Max mechanical power required in the phase [kW]
    reqm(k)= Loadm(k)/eff; % Nom mechanical power required in the phase [kW]

    HRM(k)= reqM(k)*(1-eff);% Max heat rejection the phase [kW]
    HRm(k)= reqm(k)*(1-eff);% Nom heat rejection the phase [kW]
end

if error==1
    Mass= 10^6;
else
    Mass= Pmax/PowDen;
end
```

It is important to note that this code includes a condition which checks that the generator is actually receiving mechanical power for all scenarios where it is expected to provide electric energy. If the generator is loaded without receiving mechanical power its weight will indicated a figure of one million kg (in order to indicate to the architects that the architecture concept is infeasible).

The following box represents the sizing code for the VFG:

```
function
[Mass,reqM,reqm,HRM,HRm]=VFG(DT,Cruisek,show,Vdel,LoadM,Loadm,fnom,frat,RPMnom,SpRat)


%% Model itself
% Technology factors
PowDen = 3.6;          % Power density of the generator [kg/kW]
keff = 0.95;             % Generator efficiency

% Technological constraints
K = length(Vdel);% # of scenarios
Pmax = max(LoadM); % Power capacity of the generator [kW]

error =0;% change to 1 if a requirement is infeasible
A=zeros(K,1);
reqM=A;reqm=A;HRM=A;HRm=A;

for k=1:K
    eff = keff; % This could be expressed as a function of speed and load

    if LoadM(k)>0 && RPMnom(k)==0
        error=1;
    end
    reqM(k)= LoadM(k)/eff; % Max mechanical power required in the phase [kW]
    reqm(k)= Loadm(k)/eff; % Nom mechanical power required in the phase [kW]

    HRM(k)= reqM(k)*(1-eff);% Max heat rejection the phase [kW]
    HRm(k)= reqm(k)*(1-eff);% Nom heat rejection the phase [kW]
end

if error==1
    Mass= 10^6;
else
    Mass= Pmax/PowDen;
end
```

Appendix E

Sizing the Electric Buses

E.1 General Description of the System

The "electric bus" is the name used for referring to the subsystem distributing electric power onboard the aircraft. The electric buses are composed of a bus bar for the distribution of power, and transformation devices for transforming power from AC (variable frequency or fixed frequency) to DC or vice versa. The presence of the transformation elements is conditioned by the mission of the bus. If the power received, distributed and delivered by the bus is of the same type, the transformation devices will not be necessary and therefore not implemented.

E.2 Description of the Sizing Model

The sizing model for the bus is based on a functional regression approach. The coefficients used for the sizing are described in this section.

Several efficiency coefficients were used in model. The first corresponds to the efficiencies associated with the transformation subsystems. These coefficients are presented in the following matrix. The cells in the matrix present the value used to transform from the power type in the row to the power type in the column.

Table 93: Efficiency used in bus sizing

	DC	VFAC	FFAC
DC	1 for same voltage 98% if different voltages	98%	95%
VFAC	90%	1 for same voltage 90% if different voltages	90%
FFAC	95%	95%	1 for same voltage XX% if different voltages

The power losses on the buses were calculated using efficiency factors. The efficiency factors for the buses were defined as follows:

Table 94: Bus efficiencies

Bus	Bus efficiency
DC	99%
VFAC	98%
FFAC	97%

The power density used for all bus models was 0.01 kg/kW/M. The weight of the buses was also used to highlight an inconsistency in architecture configurations. If the bus is expected to deliver energy without being connected to any source, the weight of the bus will be defaulted to 10^6 kg.

E.3 Code

E.3.1 Code for the DCBus

```
function
[Mass,ReqMDC,ReqnomDC,ReqMVFAC,ReqnomVFAC,ReqMFFAC,ReqnomFFAC]=DCBus(DT,Cruisek,show,Leng
th,Vnom,VrecDC,VrecFFAC,VrecVFAC,VdelDC,LoadMDC,LoadnomDC,VdelFFAC,LoadMFFAC,LoadnomFFAC,
VdelVFAC,LoadMVFAC,LoadnomVFAC)

%% Model itself
% Technology factors
Density = 0.01;         % Power density of the bus [kg/kW/m]

ett_DC2DC=0.98;         % Efficiency DC to DC
ett_VFAC2DC=0.90;       % Efficiency VFAC to DC
ett_FFAC2DC=0.95;       % Efficiency FFAC to DC
ett_bus = 0.99;         % Bus efficiency
ett_DC2VFAC=0.90;       % Efficiency DC to VFAC
ett_DC2FFAC=0.95;       % Efficiency DC to FFAC

% Technological constraints
K=length(VrecDC);A = zeros(1,K);
PoutM = A; Poutm = A; PinM = A; Pinm = A;
ReqMDC = A; ReqnomDC = A; ReqMVFAC = A;
ReqnomVFAC = A; ReqMFFAC = A; ReqnomFFAC = A;
Error=0;                % Test variables (should be 0 if 1 it means that
                        % the bus is asked to deliver power with no source)
for k=1:K
    if VdelDC(k)==Vnom
        ett_DC=1;
    else
        ett_DC=ett_DC2DC;
    end

    % Losses due to transformation at delivery
    PoutM(k)=1/ett_VFAC2DC*LoadMVFAC(k)+...
             1/ett_FFAC2DC*LoadMFFAC(k)+...
             1/ett_DC*LoadMDC(k);
    Poutm(k)=1/ett_VFAC2DC*LoadnomVFAC(k)+...
             1/ett_FFAC2DC*LoadnomFFAC(k)+...
             1/ett_DC*LoadnomDC(k);

    % Losses due to transmission in the bus
    PinM(k)=PoutM(k)/ett_bus;
    Pinm(k)=Poutm(k)/ett_bus;


    % Determination of the number of sources
    if PinM(k)+Pinm(k)~=0
        if VrecDC(k)+VrecFFAC(k)+VrecVFAC(k)==0
            Error=1;
        else
            ShareDC=(VrecDC(k)~=0);
            ShareVFAC=(VrecVFAC(k)~=0);
            ShareFFAC=(VrecFFAC(k)~=0);
            TotShare = ShareDC + ShareVFAC + ShareFFAC;

            if VrecDC(k)==Vnom
                ett_DC=1;
            else
                ett_DC=ett_DC2DC;
            end

            % Assignement of loads to sources
            ReqMDC(k) = ShareDC/TotShare/ett_DC*PinM(k);
            ReqnomDC(k) = ShareDC/TotShare/ett_DC*Pinm(k);
            ReqMVFAC(k) = ShareVFAC/TotShare/ett_DC2VFAC*PinM(k);
            ReqnomVFAC(k) = ShareVFAC/TotShare/ett_DC2VFAC*Pinm(k);
            ReqMFFAC(k) = ShareFFAC/TotShare/ett_DC2FFAC*PinM(k);
            ReqnomFFAC(k) = ShareFFAC/TotShare/ett_DC2FFAC*Pinm(k);
        end
    end
end
if Error==0
    Mass = max(PinM)*Density*Length;
```

```
else
    Mass = 10^6;
end
```

E.3.2 Code for the VFACBus

```
function
[Mass,ReqMDC,ReqmDC,ReqMVFAC,ReqmVFAC,ReqMFFAC,ReqmFFAC]=VFACBus(DT,Cruisek,show,Length,V
nom,VrecDC,VrecFFAC,VrecVFAC,VdelDC,LoadMDC,LoadmDC,VdelFFAC,LoadMFFAC,LoadmFFAC,VdelVFAC
,LoadMVFAC,LoadmVFAC)



%% Model itself
% Technology factors
Density = 0.01;         % Power density of the bus [kg/kW/m]

ett_DC2VFAC=0.98;         % Efficiency DC to VFAC
ett_VFAC2VFAC=0.90;       % Efficiency VFAC to VFAC
ett_FFAC2VFAC=0.95;       % Efficiency FFAC to VFAC
ett_bus = 0.98;           % Bus efficiency
ett_VFAC2DC=0.90;         % Efficiency VFAC to VFAC
ett_VFAC2FFAC=0.95;       % Efficiency VFAC to FFAC

% Technological constraints
K=length(VrecDC);A = zeros(1,K);
PoutM = A; Poutm = A; PinM = A; Pinm = A;
ReqMDC = A; ReqmDC = A; ReqMVFAC = A;
ReqmVFAC = A; ReqMFFAC = A; ReqmFFAC = A;
Error=0;                % Test variables (should be 0 if 1 it means that
                        % the bus is asked to deliver power with no source)
for k=1:K
    if VdelVFAC(k)==Vnom
        ett_VFAC=1;
    else
        ett_VFAC=ett_VFAC2VFAC;
    end

    % Losses due to transformation at delivery
    PoutM(k)=1/ett_DC2VFAC*LoadMDC(k)+...
             1/ett_FFAC2VFAC*LoadMFFAC(k)+...
             1/ett_VFAC*LoadMVFAC(k);
    Poutm(k)=1/ett_DC2VFAC*LoadmDC(k)+...
             1/ett_FFAC2VFAC*LoadmFFAC(k)+...
             1/ett_VFAC*LoadmVFAC(k);

    % Losses due to transmission in the bus
    PinM(k)=PoutM(k)/ett_bus;
    Pinm(k)=Poutm(k)/ett_bus;


    % Determination of the number of sources
    if PinM(k)+Pinm(k)~=0
        if VrecDC(k)+VrecFFAC(k)+VrecVFAC(k)==0
            Error=1;
        else
            ShareDC=(VrecDC(k)~=0);
            ShareVFAC=(VrecVFAC(k)~=0);
            ShareFFAC=(VrecFFAC(k)~=0);
            TotShare = ShareDC + ShareVFAC + ShareFFAC;

            if VrecVFAC(k)==Vnom
                ett_VFAC=1;
            else
                ett_VFAC=ett_VFAC2VFAC;
            end

            % Assignement of loads to sources
            ReqMDC(k) = ShareDC/TotShare/ett_VFAC2DC*PinM(k);
            ReqmDC(k) = ShareDC/TotShare/ett_VFAC2DC*Pinm(k);
            ReqMVFAC(k) = ShareVFAC/TotShare/ett_VFAC*PinM(k);
            ReqmVFAC(k) = ShareVFAC/TotShare/ett_VFAC*Pinm(k);
            ReqMFFAC(k) = ShareFFAC/TotShare/ett_VFAC2FFAC*PinM(k);
            ReqmFFAC(k) = ShareFFAC/TotShare/ett_VFAC2FFAC*Pinm(k);
        end
    end
end
if Error==0
```

```
    Mass = max(PinM)*Density*Length;
else
    Mass = 10^6;
end
```

E.3.3 Code for the FFACBus

```
function
[Mass,ReqMDC,ReqmDC,ReqMVFAC,ReqmVFAC,ReqMFFAC,ReqmFFAC]=FFACBus(DT,Cruisek,show,Length,V
nom,VrecDC,VrecFFAC,VrecVFAC,VdelDC,LoadMDC,LoadmDC,VdelFFAC,LoadMFFAC,LoadmFFAC,VdelVFAC
,LoadMVFAC,LoadmVFAC)

%% Model itself
% Technology factors
Density = 0.01;         % Power density of the bus [kg/kW/m]

ett_DC2FFAC=0.95;         % Efficiency DC to FFAC
ett_VFAC2FFAC=0.90;       % Efficiency VFAC to FFAC
ett_FFAC2FFAC=0.98;       % Efficiency FFAC to FFAC
ett_bus = 0.97;           % Bus efficiency
ett_FFAC2DC=0.90;         % Efficiency FFAC to VFAC
ett_FFAC2VFAC=0.95;       % Efficiency FFAC to VFAC

% Technological constraints
K=length(VrecDC);A = zeros(1,K);
PoutM = A; Poutm = A; PinM = A; Pinm = A;
ReqMDC = A; ReqmDC = A; ReqMVFAC = A;
ReqmVFAC = A; ReqMFFAC = A; ReqmFFAC = A;
Error=0;                % Test variables (should be 0 if 1 it means that
                        % the bus is asked to deliver power with no source)
for k=1:K
    if VdelFFAC(k)==Vnom
        ett_FFAC=1;
    else
        ett_FFAC=ett_FFAC2FFAC;
    end

    % Losses due to transformation at delivery
    PoutM(k)=1/ett_DC2FFAC*LoadMDC(k)+...
             1/ett_VFAC2FFAC*LoadMVFAC(k)+...
             1/ett_FFAC*LoadMFFAC(k);
    Poutm(k)=1/ett_DC2FFAC*LoadmDC(k)+...
             1/ett_VFAC2FFAC*LoadmVFAC(k)+...
             1/ett_FFAC*LoadmVFAC(k);

    % Losses due to transmission in the bus
    PinM(k)=PoutM(k)/ett_bus;
    Pinm(k)=Poutm(k)/ett_bus;


    % Determination of the number of sources
    if PinM(k)+Pinm(k)~=0
        if VrecDC(k)+VrecFFAC(k)+VrecVFAC(k)==0
            Error=1;
        else
            ShareDC=(VrecDC(k)~=0);
            ShareVFAC=(VrecVFAC(k)~=0);
            ShareFFAC=(VrecFFAC(k)~=0);
            TotShare = ShareDC + ShareVFAC + ShareFFAC;

            if VrecFFAC(k)==Vnom
                ett_FFAC=1;
            else
                ett_FFAC=ett_FFAC2FFAC;
            end

            % Assignement of loads to sources
            ReqMDC(k) = ShareDC/TotShare/ett_FFAC2DC*PinM(k);
            ReqmDC(k) = ShareDC/TotShare/ett_FFAC2DC*Pinm(k);
            ReqMFFAC(k) = ShareFFAC/TotShare/ett_FFAC*PinM(k);
            ReqmFFAC(k) = ShareFFAC/TotShare/ett_FFAC*Pinm(k);
            ReqMVFAC(k) = ShareVFAC/TotShare/ett_FFAC2VFAC*PinM(k);
            ReqmVFAC(k) = ShareVFAC/TotShare/ett_FFAC2VFAC*Pinm(k);
        end
    end
end
if Error==0
    Mass = max(PinM)*Density*Length;
```

```
else
    Mass = 10^6;
end
```

Appendix F

Battery performance and sizing model

Nomenclature for Appendix F:

DoD	Depth of discharge	R_{int}	Internal resistance
Cp	Peukert charge capacity	*S*	Step size
Δt	Time Step	*Status*[.]	Batt. status (providing/ recharging)
I	Current	*T*	Duration
K	Peukert coefficient	*T*	reference to point at time t
Loads[.]	Power requirement on batt.	*TestSize*	Constraint witness
Ncell	Number of cell in the battery	*V*	Voltage delivered by the battery
Preq	Power requirements for recharging	*Vnom*	Nominal voltage required by loads

This appendix introduces the code which was defined for the sizing process of a battery. The battery subsystem was represented as a performance model imbedded in an optimizer for sizing. The overall flow of variables is represented in the following figure:

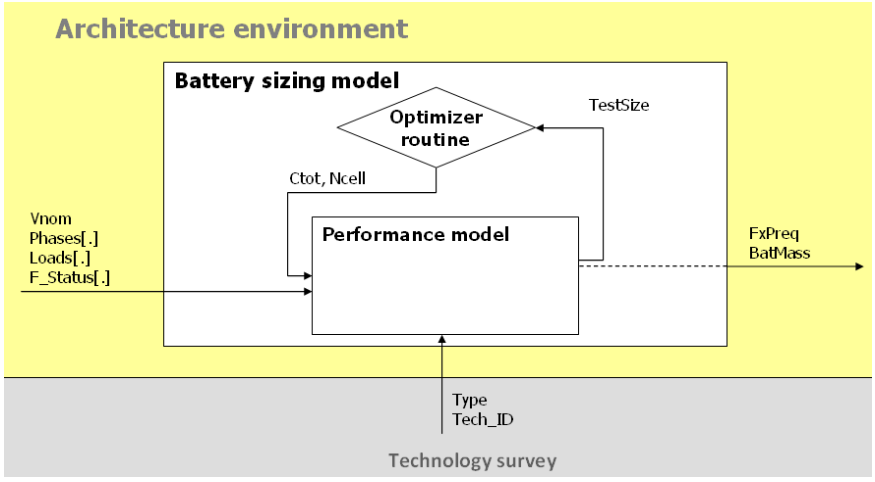


Figure 287: Battery Model Overview

This appendix will first present the physics that will be used in the sizing process of the battery. It will then present the algorithms of the model. This section will be concluded by an example and the actual code.

F.1 Battery performance model

The battery model will assume that battery can be described by the following circuit:

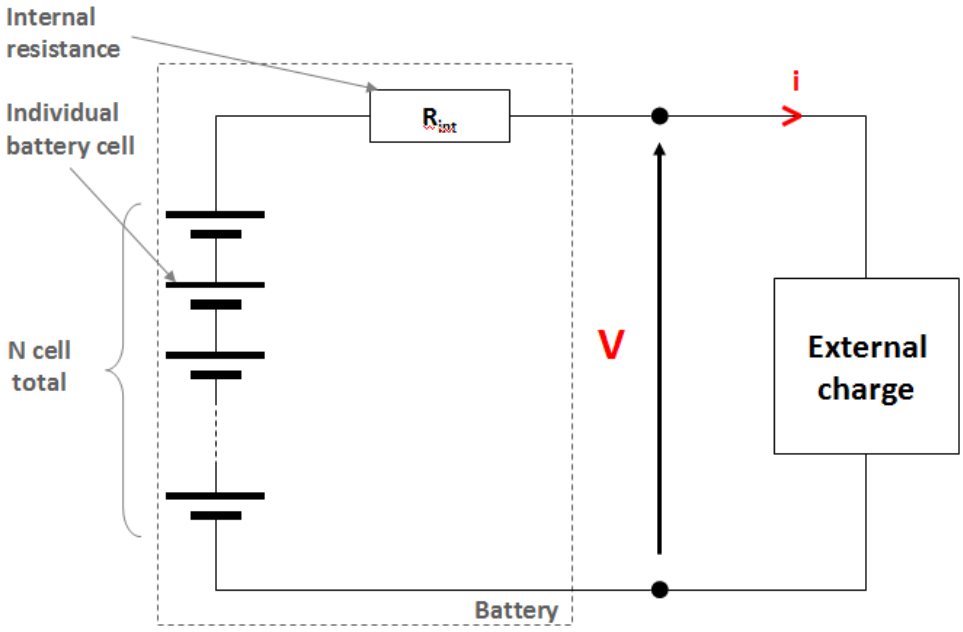


Figure 288: Representation of the battery

The each battery cell is presented as an electric source with a potential *V* from which a current *I* is drawn. Given the fact that in the architecture environment are expressed as power, the current *I* can be derived by the following equation which is a modified form of the electric power equation.

The voltage provided by the battery is degraded by internal resistance which modifies the overall potential provided by the battery. This voltage can be represented as a function of the depth of discharge. The voltage drop for a lead acid battery is represented in Figure 289.

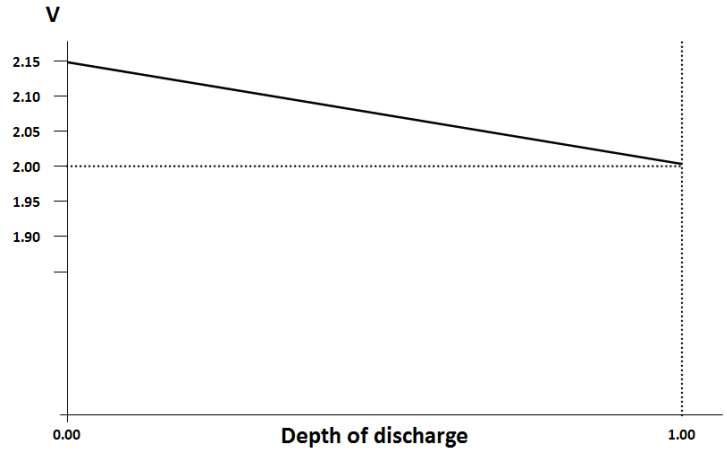


Figure 289: Typical voltage drop for a lead battery

Therefore the potential delivered by the battery is represented by the following equation where nc_{ell} is the number of cell in the battery and f is a function capturing the effect of internal resistance with repesct to the depth of discharge. The function f will be different for each battery technology.

(132)

$$V = n_{cell} \times f_{techtype}(DoD)$$

The depth of discharge is a value which represents the level of energy potential usage of a battery. When the battery is fully charged, the Depth of Discharge (DoD) is equal to zero. As current is drawn from the battery the DoD progressively increase to 1.

(133)

$$DoD(t) = \frac{C(t)}{Ctot}$$

The amount of energy present in a battery is referred to as the **charge** (C). Charge is expressed in amperes-hours (Ah). The charge of a battery which is fully charged is referred to as $Ctot$. A battery with a capacity of 1 Ah means that once fully charged the battery is capable of providing one ampere over one hour. The complexity associated with the qualification of the capacity of a battery comes from the fact that the amount of energy that it varies with the discharge rate (current drawn from the battery). In other words a battery capable of providing 1 ampere over an hour may not be capable

to provide 2 amperes over 30 minutes. Therefore capacity or charge of a battery is generally associated with specified duration.

Since the charge consumption is a function of the current drawn from the battery, the power draw must be expressed in the form of amperes rather than watts (as it is done in the architecture environment). In order to derive the current from the power of the battery load the following equation is used:

(134)

$$I = \frac{P[W]}{V}$$

In order to capture the rate of discharge of the battery, the Peukert capacity relationship was used. The Peukert capacity relationship is defined in Equation (135).

(135)

$$C_p = I^k T$$

k is referred to as the Peukert' coefficient. This coefficient is dependent on the reactive material used in the battery. *T* specifies the time during which current *I* was imposed on the battery and *Cp* is the "Peukert capacity" which defines the overall capacity of the battery (for an expected duration over one unit of time). The relationship specified in Equation (135) therefore provides a relationship between the charge of a battery, the constant current imposed by the load and the time during which the load can be provided for.

For situations where the load imposed on the battery is not constant, the charge removed can be estimated by rearranging Equation (135) into its derivative form:

(136)

$$\partial C = -I(t)^k \partial t$$

If time steps Δt are used by the model, the following equation is used to estimate evolution of the DoD.

(137)

$$DoD(t+\Delta t) = DoD(t) - \frac{I^k \Delta t}{Ctot}$$

The batteries considered in the architectures investigated in this thesis are rechargeable batteries. When a battery is recharged current is flowing in the direction opposite to the one presented in Figure 288. The recharging process will restore some charge into the battery in the form described by the following equation:

(138)

$$\partial C = I(t)\partial t$$

The speed at which the recharge process can be performed is limited by the rate at which the internal chemistry of the reactant can be performed. Based on Equation (138), the current during the charging process is limited by this recharge rate.

Each technology (type of reactant) will have its own performance recharge performance but most will be limited differently at low and high DoD levels. At high DoD (battery almost empty), the recharge process can occur at higher rate than at lower DoD.

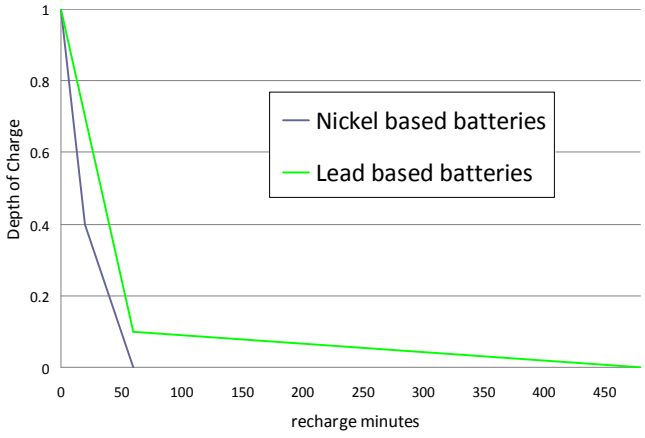


Figure 290: Recharge rates

Lead batteries:

- 90% charged in 60 min

- 100% charged in 8 hours

Nickel cadmium batteries:

- 60% charged in 20 min

- 100% charged in 60 min

To account for this variation two recharge current rates are defined.

(139)

$$I = \begin{cases} \left.\dfrac{\partial DoD}{\partial t}\right|_{fast} = \dfrac{DoD_t}{T_{trans}} & \text{for } DoD \le DoD_t \\ \left.\dfrac{\partial DoD}{\partial t}\right|_{slow} = \dfrac{1 - DoD_t}{T_{total} - T_{trans}} & \text{for } DoD > DoD_t \end{cases}$$

The performance model used in the architecture model was based on the equations (132) through (139). The flowchart for evaluating the performance of the battery is presented in Figure 291.

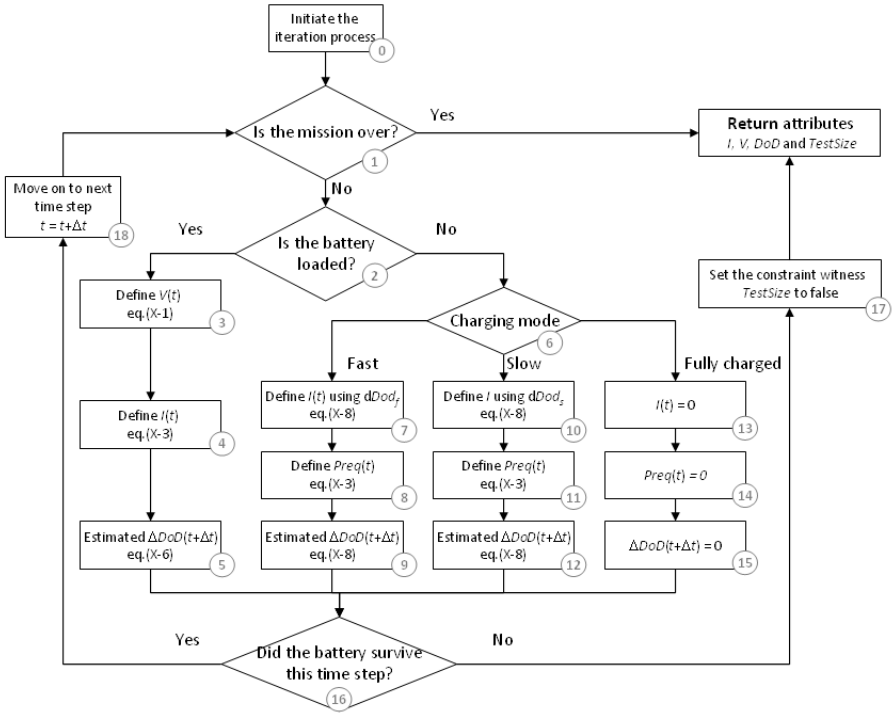


Figure 291: Flowchart for the battery performance model

F.2 Battery optimizer process

In the sizing process of the battery, the optimization is managing the size of the battery so that it is capable of providing the loads required by the architecture environment while minimizing the weight of the battery. The optimizer in this subsystem is fairly simple because it only considers one attribute (weight) in the optimization process which is dependent only on one design variable, *Ctot*.

Overall two design variables are defined by the optimizer routine:

- *Ctot*: total charge capacity of the battery.
- *Ncell*: the number of cell constituting the battery

Ncell is directly defined by *Vnom* which is the target voltage required upon the battery and the cell potential associated with the battery considered (for example a lead cell has a voltage of 2V – regardless of the overall battery size; a Nickel-Cadmium cell has a voltage of 1.2V). The equation used to determine *Ncell* is:

(140)

$$Ncell = ceiling\left[Vnom / V_{\text{per cell}} \right]$$

Ctot is defined by the iterative process presented in Figure 248. This iterative process will propose a value for *Ctot* which will be submitted to the performance model. The performance model will return the battery attribute along with a Boolean constraint witness. This Boolean *TestSize* will take the value "*true*" if the battery is able to perform the mission and "*false*" if it failed. It is to be noted that if *TestSize* is returned as "*false*" it means that the battery is undersized; therefore the battery size *Ctot* should be increased. On the other hand, if *TestSize* is returned as "*true*" it means that the battery capacity is exceeding the requirement and is possibly oversized; therefore the optimizer will attempt to decrease the value of *Ctot*.

This iterative process searching for the optimal value of *Ctot* is therefore based on a search step (used to increase or decrease the value of *Ctot*). This step size is controlled by a routine condition which will progressively decrease its amplitude as the routine is passing around the optimal value.

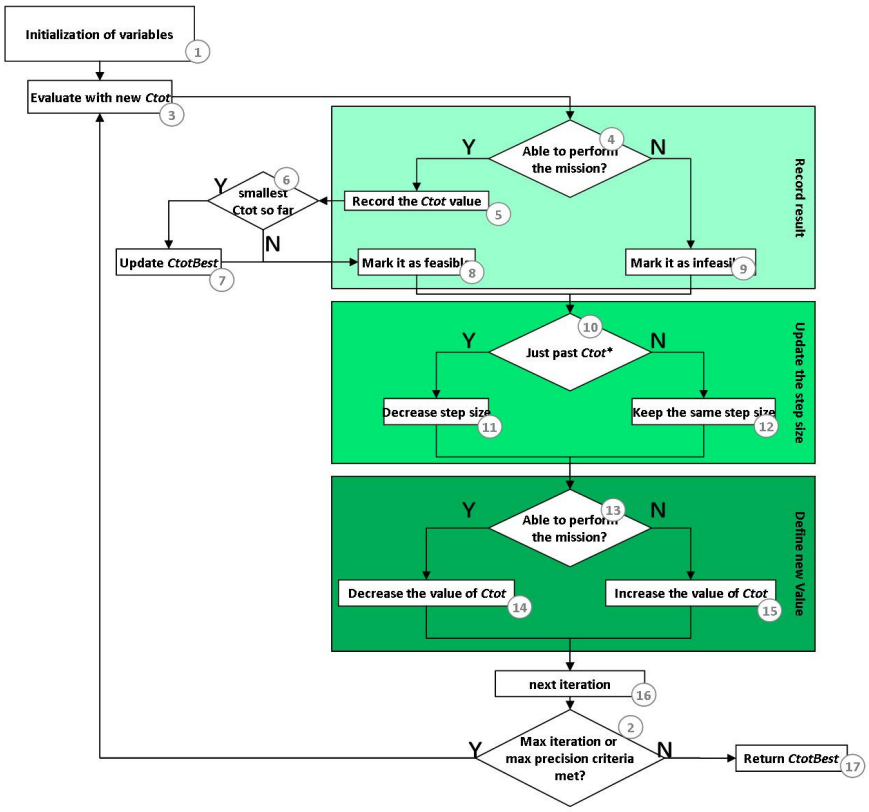


Figure 292: Battery optimizer flowchart

F.3 Illustration and example

To illustrate the capability of the sizing model for the battery the following load profile was used:

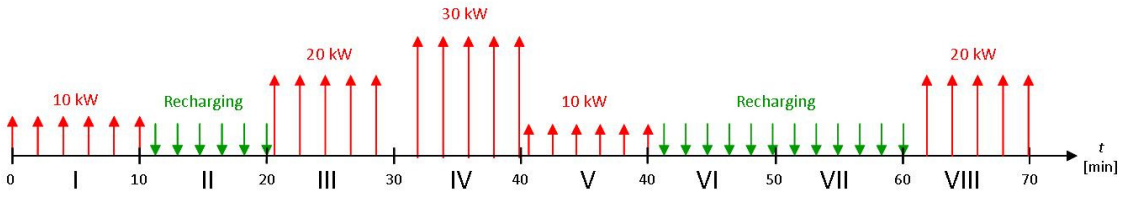


Figure 293: Example load profile

To represent this mission the routine will be receiving three vectors:

- A vector describing the duration of the 8 phases above: [10,10,10, 5, 5,20,10]
- A vector specifying the functional status of the battery in each phase: [1 ,-1,1 ,1 ,1 ,-1,1] (Note: 1 – Battery is loaded, -1 – the battery is recharged and 0 – Mark for the initiation of a new mission segment)
- A vector quantifying the amount of power drawn from the battery in each phase: 1000*[10,0 ,20,30,10,0 ,10]

This input is received by a subroutine which is going to transform these phase based vectors into time-based vectors with a given time step. For this example the Δt was set at 1 min.

After the transformation operated by the subroutine presented in the code section of this appendix, each of the vectors above are redefined with T elements, where T is the total number of time steps in the mission.

In order to illustrate the analysis of the performance model of the battery, we are going to observe the performance profiles provided by the model for 3 Nickel-Cadmium 320V batteries. The first one will be optimal for the mission specified in Figure 293, the second will be undersized for the mission, and the third will be oversized.

F.3.1 Observation of the output of an optimal battery

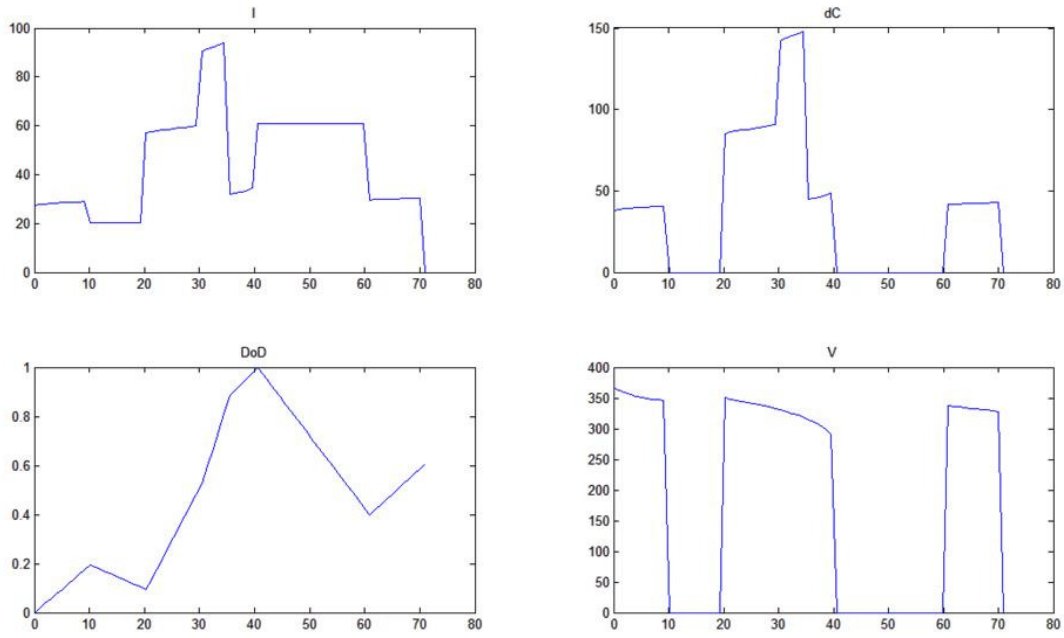


Figure 294: Performance of an optimal battery

In each of the plots above the horizontal axis represents time in minutes. In the upper-left quadrant we can see the evolution of the current on the battery. In each flight phase, we can also observe the there is a slight increase of the current with time which corresponds to the fact that the Voltage (shown in lower right corner) is drooping as the DoD is progressively decreasing as energy is delivered. The charge delivered per minute is presented in the upper-right plot. The evolution of this metric shares similar feature with *I*'s. But compared to *I,* the amplitude in the discharge rates is exacerbated by the Peukert coefficient *k* presented in equation (136). We can also observe that the charging process is captured. In phases spanning between t = 10 - 20 and 40 – 60 the DoD is progressively decreasing. It is also important to observe that the two charge rates are captured. Unlike the second recharging phase, the first has too shallow of a DoD to allow for the rapid recharge rate.

The third plot is the one that is primarily considered in the sizing process of the battery. It corresponds to the DoD (depth of discharge). As the battery is used, the DoD is

progressively increasing to the value of 1 (full discharge). At that point, the battery is not capable of providing energy. This battery is optimal because it reaches DoD=1 at the end of a series of energy delivering phases (spanning between *t*=0 and 40 minutes). It is also interesting to observe that in the mission observed in this example, the sizing sequence corresponds to phases I-V.

F.3.2 Output corresponding to suboptimal batteries

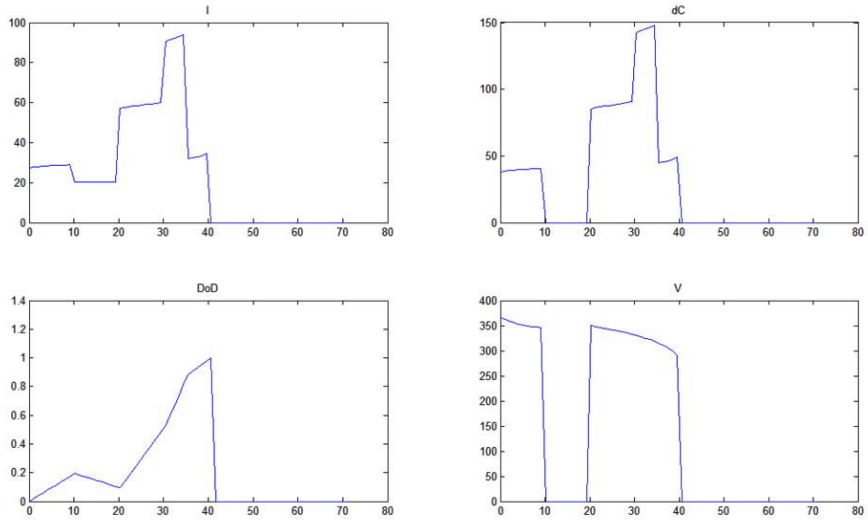


Figure 295: Performance of an undersized battery

If the battery is undersized it will reach a DoD before the end of an energy delivering phase. In the example illustrated above, full discharge is reached 1 minute before the end of phase V. At that point the simulation will stop and will return a warning with the constraint witness *TestSize*.

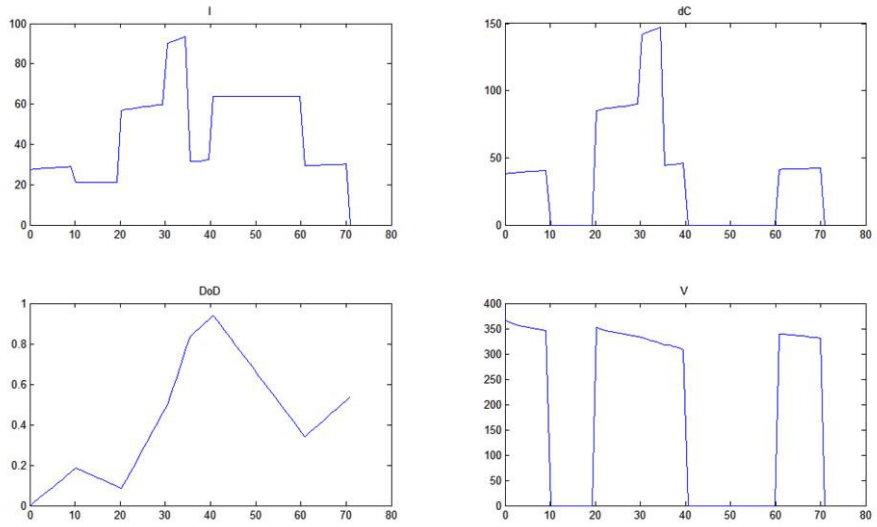


Figure 296: Performance of an oversized battery

The oversized battery will be capable of performing the mission but will be larger than necessary. In this situation, the DoD will never be complete. Therefore the battery size can be reduced without reducing its capacity to perform the mission.

F.3.3 Model accuracy and validation

No physical data were used to calibrate this model. The physical meaning of the equations used in its definition is widely accepted. Therefore in order to accurately represent the current state of the art in battery design, this model should be provided with validated Peukert coefficients, voltage relationships with DoD and recharge rates.

The relationships and value used in this model were provided in references [96].

F.4 Code

In order to make the code understandable by the reader, a reference to the flow charts in Figure 291 and Figure 292 was used to document the code. These references are recognizable by their "%-##" format. The "##" make reference to the numbered references of the steps in the flowcharts.

F.4.1 Optimizer routine:

```
function
[CtotBest]=BattOptimizerRoutine(Load,Status,Ncell,Type,Tech_ID,Dt,Vnom,OptSettings)

Sinit=OptSettings(1);
Ctotinit=OptSettings(2);
MaxIt = OptSettings(3);
Sacc = OptSettings(4); %Expected accuracy for convergence test

%-01
CtotBest = 0;
it = 1;Scount=0;
Ctot = zeros(MaxIt,1);
S = zeros(MaxIt,1);
Test = zeros(MaxIt,1);
S(1) = Sinit;   %Initial step (must be large in order for the routine to
% adapt both to small and large loads)
Ctot(1) = Ctotinit;  %Initial guess
%-02
while (it<MaxIt)
    if (S(it)>Sacc) || (CtotBest==0)
        %-03
        [TestSize,V,I,dC,DoD,Pd,k] = EvaluateBatCapacitytoMission(...
            Ctot(it),Ncell,Load,Status,Type,Tech_ID,Dt,Vnom,0);
         %-04
        if TestSize
            %-05
            Scount= Scount+1;
            CtotS(Scount)=Ctot(it);
            %-06
            if Ctot(it)<CtotBest || CtotBest == 0
                %-07
                CtotBest = Ctot(it);
            end
            %-08
            Test(it) = 1;
        else
            %-09
            Test(it) = -1;
        end
        %-10
        if it~=1 && Test(it)*Test(it-1)<0
            %-11
            S(it+1) = S(it)/10;
        else
            S(it+1) = S(it);
        end
        %-13
        if TestSize
            %-14
            Ctot(it+1) = Ctot(it) - S(it+1);
        else
            %-15
            Ctot(it+1) = Ctot(it) + S(it+1);
        end
        %16
    end
    it=it+1;
end
 %-17
%Return is explicit
```

F.4.2 Battery performance model:

```
function                                [TestSize,V,I,dC,DoD,Pd,k]                                  =
BattPerfModel(Ctot,Int_n,Load,F_Status,Type,Tech_ID,DT,Arch_Vnom,Plottest)
%-00
K = length(F_Status);
V = zeros(1,K);
I = zeros(1,K);
dC= zeros(1,K);
DoD = zeros(1,K);
Pd = zeros(1,K);

k=1;    %Rank of the scenario considered
        %Note rank 1 corresponds to t=0 and n to t = (n-1)*Dt
TestSize = (DoD(k)<1); % This condition is checked at every iteration

%-01
while k<K && TestSize
    %-02
    if F_Status(k)==1 %The battery is loaded
        %-03
        %Definition of the open circuit voltage for each cell
        if (Type == 1)
            EC = (2.15 - ((2.15-2.00)*DoD(k)));
        else
            EC      =      (-8.2816*DoD(k)^7+23.5749*DoD(k)^6-30*DoD(k)^5+23.7053*DoD(k)^4-
12.5877*DoD(k)^3+4.1315*DoD(k)^2-0.8658*DoD(k)+1.37);
        end
        V(k) = Int_n*EC;                %Voltage in circuit
        %-04
        I(k) = Load(k)/V(k);            %Current in circuit
        % It(k)= (V(k)-sqrt(V(k)^2-4*Int_n*Load(k)*0.022/Ctot))/(2*Int_n*0.022/Ctot);
        % This model will assume that internal resistance beyond open
        % voltage drop is negligeable
        %-05
        dC(k)= I(k)^Tech_ID(2,Type)*DT;    %Loss of charge
        DoD(k+1) = DoD(k) + dC(k)/Ctot; %Update depth of discharge

    %-02
    elseif F_Status(k)==-1 %The battery is charged
        DoDt = Tech_ID(3,Type);         %DoD at which the batt start slow charge rate
        dDoDf = Tech_ID(4,Type);         %Fast recharge rate
        dDoDs = Tech_ID(5,Type);         %Fast recharge rate
        %-06
        if DoD(k) > DoDt %Fast recharge
            %-07
            I(k) = dDoDf*Ctot;          %Evaluation of current during recharge
            %-08
            Pd(k)  =  I(k)*Arch_Vnom;                   %Evaluation  of  power  requiered  by
recharging process
            %-09
            DoD(k+1) = DoD(k) - dDoDf*DT;%Update depth of discharge
        %-06
        elseif DoD(k) > 0 %Slow recharge
            %-10
            I(k) = dDoDs*Ctot;          %Evaluation of current during recharge
            %-11
            Pd(k)  =  I(k)*Arch_Vnom;                   %Evaluation  of  power  requiered  by
recharging process
            %-12
            DoD(k+1) = DoD(k) - dDoDs*DT;%Update depth of discharge
        %-06
        else %The battery was already charged
            %-13
            I(k) = 0;
            %-14
            Pd(k) =0;           %No power requirements
            %-15
            DoD(k+1)= 0;
        end
        %-16
        if DoD(k+1)< 0
            DoD(k+1)=0;
        end
    elseif F_Status(k)==0 %This is a dummy scenario to reinitialize the battery
                          % i.e. This point starts a new mission sequence
        DoD(k+1)=0; %The DoD is reinitialized (fully recharged at park station
        I(k) = 0;
```

```
        Pd(k) =0;           %No power requirements
    end

    %-18
    k = k + 1;
    %-16 & 17
    TestSize = (DoD(k)<1);

end

if Plottest == 1
    Totaltime = length(I)*DT;
    Time=linspace(0,Totaltime,length(I));
    subplot(2,2,1);plot(Time,I);title('I')
    subplot(2,2,2);plot(Time,dC);title('dC')
    subplot(2,2,3);plot(Time,DoD);title('DoD')
    subplot(2,2,4);plot(Time,V);title('V')
end
```

F.4.3 Sizing model

```
clc
clear all
close all
%% Inputs

%Arch variables
Vnom = 320;%[V]

%Analysis settings
Sinit=15000;% Initial value for search step size
Ctotinit=0.1;%Initial guess for Ctot
MaxIt = 100;%Max number of iterations
Sacc = 0.1; %Expected accuracy for convergence test
OptSettings=[Sinit,Ctotinit,MaxIt,Sacc];


%Tech
Type = 2; % 1 LeadAcid, 2 NiCad
Tech_ID(1,:)= [2,1.2];  %Voltage per cell [V]
Tech_ID(2,:)= [1.107,1.1];%Peukert exponent [-]
Tech_ID(3,:)= [0.1,0.2];%Minimum DoD level for fast recharge
Tech_ID(4,:)= [0.015,0.03];%Fast recharge rate [/min]
Tech_ID(5,:)= [2.38e-4,0.01];%Slow recharge rate [/min]

% Building the mission inputs
Dt = 1;%Time step[min]
F_Status =  [1 ,-1,1 ,1 ,1 ,-1,1 ,0,1 ,-1,1 ,1 ,-1,1 ]; %Fctnal status
                                                        %[-1:   rech.,1:   loaded,   0:   New
segment]
Loads_Bat = [10,0 ,20,30,10,0 ,10,0,10,0 ,20,10,0 ,30]*1000; %Load for each segment [W]
Phases =    [10,10,10, 5, 5,20,10,5,10,10,10, 5,20,10]; %Duration of each segment [min]

[Load]=SetupTimeSteps(Dt,Loads_Bat,Phases);
[Status]=SetupTimeSteps(Dt,F_Status,Phases);

%% Solving for fixed Design Variables (number of battery cells)
VoltagePerCell = Tech_ID(1,Type);
Ncell = ceil(Vnom/VoltagePerCell);

%% Launching optimization routine
[CtotBest]=BattOptimizerRoutine(Load,Status,Ncell,Type,Tech_ID,Dt,Vnom,OptSettings);

%% Evaluation of battery's mass

Jb = CtotBest/60*Vnom; % charge capacity in W.h
if Type == 1
    BatMass = Jb/30 ;   %Value from Larminie Table 2.11
else
    BatMass = Jb/35 ;   %Value from Larminie Table 2.11
end

figure(1)
[TestSize,V,I,dC,DoD,Pd,k] = BattPerfModel(CtotBest,Ncell...
    ,Load,Status,Type,Tech_ID,Dt,Vnom,1);

FxPreq = ReturnToPhaseSteps(Dt,Pd,Phases);
```

F.4.4 Transformation between time-based and phase-based

From time to phase:

```
function [PhaseBased]=ReturnToPhaseSteps(Dt,TimeBased,Phases)
%Description of the function
% This function recieves a timestep based array. For each phase a duration
% is specified. Using the duration and a timestep size, the function will
% create a matrix where elements on row 1 return the average value over the
% the phase and on row 2 the maximum value over the phase.
% mission.

%% For testing
% clc
% clear all
% close all
% Dt = 1;%Time step[min]
% TimeBased = [10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,...
%     0,0,0,0,0,0,0,0,0,0,...
%     20000,20000,20000,20000,20000,20000,20000,20000,20000,20000,...
%     30000,30000,30000,31000,30000,10000,10000,10000,10000,10000,...
%     0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,...
%     10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,0;];
% Phases =    [10,10,10, 5, 5,20,10]; %Duration of each segment [min]
%

%% Code
TimeBrakets = zeros(1,length(Phases));
for j=1:1:length(Phases)
    for ph=1:1:j
        TimeBrakets(j)=TimeBrakets(j)+Phases(ph);
    end
end
TSteps=length(TimeBased);
PhaseInstances=zeros(1,length(Phases));
PhaseBased = zeros(2,length(Phases));
for t=1:1:TSteps
    Time= (t-1)*Dt; %Time corresponding to instance t
    for ph=1:1:length(Phases)
        if ph==1
            if Time<TimeBrakets(ph)
                PhaseBased(1,ph)=(PhaseBased(1,ph)*PhaseInstances(ph)+TimeBased(t))...
                    /(PhaseInstances(ph)+1);
                PhaseInstances(ph)=PhaseInstances(ph)+1;
                if TimeBased(t)>PhaseBased(ph,2)
                    PhaseBased(2,ph)=TimeBased(t);
                end
            end
        else
            if TimeBrakets(ph-1)<=Time && Time<TimeBrakets(ph)
                PhaseBased(1,ph)=(PhaseBased(1,ph)*PhaseInstances(ph)+TimeBased(t))...
                    /(PhaseInstances(ph)+1);
                PhaseInstances(ph)=PhaseInstances(ph)+1;
                if TimeBased(t)>PhaseBased(2,ph)
                    PhaseBased(2,ph)=TimeBased(t);
                end
            end
        end
    end
end
```

From phase to time:

```
function [TimeBased]=SetupTimeSteps(Dt,PhaseBased,Phases)
%Description of the function
% This function recieves a phase based array. For each phase a duration is
% specified. Using the duration and a timestep size, the function will
% create an arrays where each element correspond to a time step in the
% mission.

%% For testing
% clc
% clear all
% close all
% Dt = .3;%Time step[min]
% PhaseBased = [10,0 ,20,30,10,0 ,10]* 1000; %Load for each segment [W]
% Phases =    [10,10,10, 5, 5,20,10]; %Duration of each segment [min]
%

%% Code
TimeBrakets = zeros(1,length(Phases));
for j=1:1:length(Phases)
    for ph=1:1:j
        TimeBrakets(j)=TimeBrakets(j)+Phases(ph);
    end
end

TotTimesteps = 1 + ceil(sum(Phases)/Dt);
TimeBased = zeros(1,TotTimesteps);

for t=1:1:TotTimesteps
    ph=length(Phases);
    while ph>0
        if (t<=floor(TimeBrakets(ph)/Dt))
            TimeBased(t) = PhaseBased(ph);
        end
        ph=ph-1;
    end
end

Time=zeros(1,length(TimeBased));
for t=1:1:length(TimeBased)
    Time(t)=t*Dt;
end
```

Appendix G

Sizing scenario compacters

G.1 Purpose

The methodology used in this thesis performs the sizing of subsystems based on scenarios where each scenario implies a constraint. These scenarios are sometimes redundant. Therefore to accelerate the sizing process it is possible to avoid the calculators to evaluate the performance of the subsystem for the same scenario multiple times. In order to do so a compacting routine was defined to eliminate these redundant calculations. This routine [compacter] will identify the non-redundant set of scenarios implied by the mission and will store how the actual scenario implied by the mission map to this set. Once the subsystem is sized the non-redundant set is transformed back into the original set of scenario implied by the mission.

G.2 Compacter

To better understand this process, let us consider the sizing of a power device which has to provide some torque and some given speed. The quantification of the functional requirements imposed on this device require that 2 elements of information are passed for each scenario (Torque and speed). Let us assume that the mission contains 15 scenarios. The functional requirements for this mission are defined by the following table:

Table 95: Example with redundant scenario

Scenario	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Torque	1	4	2	3	5	2	3	5	2	3	2	3	2	3	2
RPM	500	800	200	100	300	200	100	300	200	100	200	100	200	100	200

In this table we can see that several scenarios are redundant and can be eliminated to simplify the sizing analysis. The compacter routine will reduce this table to:

Table 96: Example reduced

Scenario	1*	2*	3*	4*	5*
Torque	1	4	2	3	5
RPM	500	800	200	100	300

Based on this reduction, instead of running the sizing analysis on 15 scenarios only 5 scenarios should be considered to evaluate the capabilities of the subsystem. In order to relate the actual scenarios (those listed in Table 95) to those composing the reduced set, an address matrix is also defined:

Table 97: example of "address" matrix

	Scenarios in reduced set				
Orginal Scenarios	1	2	3	4	5
	0	0	6	7	8
	0	0	9	10	0
	0	0	11	12	0
	0	0	13	14	0
	0	0	15	0	0

Each column represents a scenario in the reduced set. The numbers in the columns list the orginal scenarios corresponding to each scenario in the reduced set. In this example we can see that scenario 3 in Table 96, represents scenarios 3, 6, 9, 11, 13 and 15 in Table 95.

In order to build these tables the following routine was built:

```
function [Compact,Address]=Compacter(ScenarioIn)
%% Inputs:
%Compact: table containing compacted information about the scenario
%       (redundant scenarios are not repeated)
%Address: table containing the addresses of the scenarios in the table
%input. Example if Address(x,c) = k then the information concerning
%scenario k are stored in column c of Compact.
%% Outputs:
% ScenarioIn: This table contains the information classified by scenario

TestingMode = 0;
if TestingMode ~=0
    clc
    clear all

    ScenarioIn= [1   2   4   8   6   4   9   5   7   9   6   4   7   1   2   4   8   6
                 5   3   0   6   98  9   5   9   7   0   5   8   6   1   35  4   4   5 ];
end
% K is the total number of scenarios
% N is the number of variables per scenario
% R is the number of redundant scenarios
% NM: "Never mind"
[N,K]= size(ScenarioIn);

c = 1;
TransferT = zeros(size(ScenarioIn));
Assignments = zeros(K+1,K);
C=0;
for k=1:K
    c=1;
```

```
    while c<=C
        n=1;
        while n<=N
            if ScenarioIn(n,k)==TransferT(n,c) % Up to now the column are identical
                n = n+1; % move on to next row
            else                              % The are different
                n=N+2;   % we exit the loop (N+2 is a signal)
            end
        end
        if n==N+2   % column c is not a match to column k
            c = c+1;% move on to column c+1
        else        % column c matched k
            nt=Assignments(1,c)+1; % Number of times the column has been matched
            Assignments(1,c)=nt;
            Assignments(nt+1,c)=k;
             c= C+2;
        end
    end
    if c==C+1   % Not match were found
        C=C+1;  %add a new column
        TransferT(:,C)=ScenarioIn(:,k);% Transfer the current scenario in the column
        Assignments(1,C)= 1;  % Records the fact that 1 scenario matches this column
        Assignments(1+1,C)=k; % This specifies that scenario k has been stored in column
C
    end
end
% Detecting the length of the short list of scneario (stored in Kshort):
Kshort= 0;
for k=1:K
    if sum(TransferT(:,k))~=0
        Kshort = k;
    end
end
R=max(Assignments(1,:));
Compact= TransferT(1:N,1:Kshort);
Address = Assignments(2:1+R,1:Kshort);
```

G.3 Decompacter

Once the subsystem has been sized, it is necessary to classify the attributes of the subsystem associated with each scenario. In the power device example used previously this information maybe the amount of power requirement induced by the machine in each scenario. Let us assume that the sizing of the device has led to the evaluation of the functional requirements listed in the following table:

Table 98: Example of induced requirements

Scenario	1*	2*	3*	4*	5*
Power Req.	625	4000	500	375	1875

Using the "address" matrix listed in Table 97, we can list the power requirement associated with the original scenarios:

Table 99: Example of de-compacted functional requirements

Scenario	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Power Req.	625	4000	500	375	1875	500	375	1875	500	375	500	375	500	375	500

The transformation is performed by the routine "decompacter".

```
function ScenarioOut=Decompacter(Compact,Address)
%% Inputs:
%Compact: table containing compacted information about the scenario
%       (redundant scenarios are not repeated)
%Address: table containing the addresses of the scenarios in the table
%input. Example if Address(x,c) = k then the information concerning
%scenario k are stored in column c of Compact.
%% Outputs:
% ScenarioOut: This table contains the information classified by scenario

TestingMode = 0;

if TestingMode ~=0
    clc
    clear all

    Compact = [625  4000    500 375 1875];

    Address = [ 1     2     3     4     5
                0     0     6     7     8
                0     0     9    10     0
                0     0    11    12     0
                0     0    13    14     0
                0     0    15     0     0];
end

% K is the total number of scenarios
% N is the number of variables per scenario
% R is the number of redundant scenarios
% NM: "Never mind"
K = max(max(Address));
[R,NM]=size(Address);
[N,C] = size(Compact);
ScenarioOut = zeros(N,K);

% These loops scan the address table and assign the data in Compact in a
% table classified by scenario
for c=1:C
    for r=1:R
        k = Address(r,c);
        if k~=0
            ScenarioOut(:,k) = Compact(:,c);
        end
    end
end
```

Appendix H

Scriptwrapper Generator

The sizing models for the subsystems were implemented in Matlab. In order to be able to integrate and use these models in ModelCenter (MC), a ScriptWrapper (SW) must be created. This file is a script encoded in VBscript. The original template was provided by Nathan Sharp from Phoenix integration. The creation of the SW was fairly systematic but the syntax necessary for the script to operate normally was very specific. Hence the implementation of the script was generally a tricky operation to perform by hand, especially since during the debugging and adjustment phases required adding, removing and adjusting variables names and model reference.

Therefore in order to save time and spare the team from tremendous debugging frustrations, the Scriptwrapper generator was implemented. This generator creates the SW automatically based on a simple input file including:

- The name of the Matlab function to wrap
- The name of the MC model to be created
- A list of the input variables
- A list of the output variables
- The destination folder in which the SWF must be saved

The Scriptwrapper generator was created with the assistance of Charles Nespoulous, Graduate Research Assistant at the Aerospace Systems Design Laboratory. In order to illustrate the script output of the generator, the script automatically generated for the electric fan is presented in the second section.

H.1 Script Wrapper Generator

```
function  []=ScriptWrapperGenerator(ModelName,MLfctName,Inputs,Outputs,filePath)

s = 1;% For scalar type
v = 2;% For vector type
t = '\n   ';
Testingmode=0;      % should be 0 unless you are testing the function
if Testingmode~=0

    % Step 1: Give your MC model a name
    ModelName = 'TurboElecFan';
    % Step 2: Specify the ML model name (make sure your function is on the ML path)
    MLfctName = ModelName;
    % Step 3: Define the inputs in your function
    % 3.1: the variables have to be listed in the same order as they show in
    %       your function declaration.
    % 3.2: for each specify if the variable is a scalar or a vector
    Inputs = 'DT',v,[1,1,1,1,1,1,1,1,1,1,1]
    'Cruisek',s,7
    'show',s,0
    'gamma',v,[.25,.25,.25,.25]
    'Alt',v,[0   0    0         0         50        275       350       150       100       100
0]*100
    'MN',v,[0   0   0.04    0.04    0.41    0.75    0.78    0.71    0.6     0.45    0.3]
    'ThMax',v,[0    0     100      307      213      70.5     67.6     71.8     84.2     78.2
254]*1000/8
    'ThNom',v,[ 0    0    5         236      160      77       38       0         0         0
0]*1000/8
    'Vrec',v,[1,1,1,1,1,1,1,1,1,1,1]*2*10^5};

    % Step 4: List the outputs (same as step 3)
    Outputs = {'W',s
        'ReqM',v
        'Reqnom',v
        'DV_fan',v};

    % Step 5: Specify the folder in which the scriptWrapper must be stored
    filePath='C:\Program         Files\Phoenix         Integration\Analysis         Server
5.1\analyses\Archsetup\Builder Library\';
end
nInputs=size(Inputs);
nOutputs=size(Outputs);

%% Declaration of variables
Declaration='';% Contains code declaring the MC variables (both input and output
MLInput = '(';
MLOutput= '[';
InitializeVectors ='\n';% Contains code providing default values for vector variables
setToML = [t];
readFromML= ['\n',t];
for n=1:nInputs(1)
    varName = cell2mat(Inputs(n,1));
    if cell2mat(Inputs(n,2))==s
        type = 'double';
        setToML=[setToML,'ml.PutWorkspaceData            "',varName,'",            "base",
',varName,'.value\n',t];

        defaultVal = cell2mat(Inputs(n,3));
        defaultSet=[' default=',num2str(defaultVal)];
    else
        type = 'double[]';
        setToML=[setToML,'ml.PutWorkspaceData            "',varName,'",            "base",
',varName,'.getArray()',t];
        setToML=[setToML,'ml.Execute  "',varName,'=cell2mat(',varName,')"  ''correct  for
cell array\n',t];

        defaultVal = cell2mat(Inputs(n,3));
        K=length(defaultVal);
        defaultSet='';

        InitializeVectors = [InitializeVectors,varName,'.setArray Array('];
        for k=1:K
            InitializeVectors = [InitializeVectors,num2str(defaultVal(k))];
            if k<K
                Next = ',';
            else
```

```
                Next=')\n';
            end
            InitializeVectors = [InitializeVectors,Next];
        end
    end
    Declaration=[Declaration,'variable: ',varName,' ',type,' input',defaultSet,'\n'];
    MLInput=[MLInput,varName];
    if n<nInputs(1)
        MLInput=[MLInput,','];
    else
        MLInput=[MLInput,')'];
    end
end
for n=1:nOutputs(1)
    varName = cell2mat(Outputs(n,1));
    if cell2mat(Outputs(n,2))==s
        type = 'double';
        readFromML = [readFromML,'ml.GetWorkspaceData "',varName,'", "base", temp',t,...
                      varName,'.value = temp',t,t];
    else
        type = 'double[]';
        readFromML = [readFromML,'ml.Execute "lgth = length(',varName,')"',t,...
                      'ml.Execute     "for     i     =     1:lgth     ',varName,'1(i)     =
',varName,'(i);end;"',t,...
                      'ml.GetWorkspaceData "',varName,'1", "base", temp',t,...
                      varName,'.setArray temp',t,...
                      'ml.GetWorkspaceData "lgth", "base", lgth',t,...
                      'ReDim tps',varName,'(0)',t,...
                      'tps',varName,'(0) = ',varName,'(0,0)',t,...
                      'for j = 1 to (lgth-1)',t,...
                      'ReDim preserve tps',varName,'(ubound(tps',varName,') + 1)',t,...
                      'tps',varName,'(j) = ',varName,'(0,j)',t,...
                      'next',t,...
                      varName,'.setArray tps',varName,t,t];
    end
    Declaration=[Declaration,'variable: ',varName,' ',type,' output\n'];
    MLOutput=[MLOutput,varName];
    if n<nOutputs(1)
        MLOutput=[MLOutput,','];
    else
        MLOutput=[MLOutput,']'];
    end
end

%% Comment line
c = clock;
Time=[num2str(c(3)),'/',num2str(c(2)),'/',num2str(c(1)),'                                    at
',num2str(c(4)),':',num2str(c(5)),':',num2str(floor(c(6)))];
Commentline=['''This   script   was   automatically   using   the   ScriptWrapperGenerator
function',...
    '\n''',Time,'\n'];


%% Code for ML initiation
MLinitiation =['\n',...
    'script:\n',...
    Commentline,...
    InitializeVectors,... % Set default values for the input vectors
    '\n''Launch the matlab session\n',...
    'set ml = CreateObject("Matlab.Application")\n'];

%% Code for sub
SubCode = ['\nsub run()',t,...
   'ml.Execute "clear all"',t,t,...
   '''set inputs',setToML,...           % Inserting code for input export to ML
   '''Run the ML function',t,...
   'Dim lgth',t,...
   '   ml.Execute "',MLOutput,'=',MLfctName,MLInput,';"',t,t,...% Model execution
   ''' Find the output',t,...
   readFromML,'\n',...                  % Import from ML
   'end sub'];

%% Writting to file
Code = [Declaration,MLinitiation,SubCode];

FileName = [filePath,ModelName,'.scriptWrapper'];
fid = fopen(FileName,'wt');
fprintf(fid,Code);
fclose(fid);
end
```

H.2 Illustration of automatically Generated Script

```
variable: DT double[] input
variable: Cruisek double input default=7
variable: show double input default=0
variable: gamma double[] input
variable: Alt double[] input
variable: MN double[] input
variable: ThMax double[] input
variable: ThNom double[] input
variable: Vrec double[] input
variable: W double output
variable: ReqM double[] output
variable: Reqnom double[] output
variable: DV_fan double[] output

script:
'This script was automatically using the ScriptWrapperGenerator function
'23/3/2010 at 17:7:22

DT.setArray Array(1,1,1,1,1,1,1,1,1,1,1)
gamma.setArray Array(0.25,0.25,0.25,0.25)
Alt.setArray Array(0,0,0,0,5000,27500,35000,15000,10000,10000,0)
MN.setArray Array(0,0,0.04,0.04,0.41,0.75,0.78,0.71,0.6,0.45,0.3)
ThMax.setArray Array(0,0,12500,38375,26625,8812.5,8450,8975,10525,9775,31750)
ThNom.setArray Array(0,0,625,29500,20000,9625,4750,0,0,0,0)
Vrec.setArray
Array(200000,200000,200000,200000,200000,200000,200000,200000,200000,200000,200000)

'Launch the matlab session
set ml = CreateObject("Matlab.Application")

sub run()
   ml.Execute "clear all"

   'set inputs
   ml.PutWorkspaceData "DT", "base", DT.getArray()
   ml.Execute "DT=cell2mat(DT)" 'correct for cell array

   ml.PutWorkspaceData "Cruisek", "base", Cruisek.value

   ml.PutWorkspaceData "show", "base", show.value

   ml.PutWorkspaceData "gamma", "base", gamma.getArray()
   ml.Execute "gamma=cell2mat(gamma)" 'correct for cell array

   ml.PutWorkspaceData "Alt", "base", Alt.getArray()
   ml.Execute "Alt=cell2mat(Alt)" 'correct for cell array

   ml.PutWorkspaceData "MN", "base", MN.getArray()
   ml.Execute "MN=cell2mat(MN)" 'correct for cell array

   ml.PutWorkspaceData "ThMax", "base", ThMax.getArray()
   ml.Execute "ThMax=cell2mat(ThMax)" 'correct for cell array

   ml.PutWorkspaceData "ThNom", "base", ThNom.getArray()
   ml.Execute "ThNom=cell2mat(ThNom)" 'correct for cell array

   ml.PutWorkspaceData "Vrec", "base", Vrec.getArray()
   ml.Execute "Vrec=cell2mat(Vrec)" 'correct for cell array

   'Run the ML function
   Dim lgth
      ml.Execute
"[W,ReqM,Reqnom,DV_fan]=TurboElecFan(DT,Cruisek,show,gamma,Alt,MN,ThMax,ThNom,Vrec);"

   ' Find the output


   ml.GetWorkspaceData "W", "base", temp
   W.value = temp

   ml.Execute "lgth = length(ReqM)"
   ml.Execute "for i = 1:lgth ReqM1(i) = ReqM(i);end;"
   ml.GetWorkspaceData "ReqM1", "base", temp
   ReqM.setArray temp
   ml.GetWorkspaceData "lgth", "base", lgth
   ReDim tpsReqM(0)
```

```
    tpsReqM(0) = ReqM(0,0)
    for j = 1 to (lgth-1)
    ReDim preserve tpsReqM(ubound(tpsReqM) + 1)
    tpsReqM(j) = ReqM(0,j)
    next
    ReqM.setArray tpsReqM

    ml.Execute "lgth = length(Reqnom)"
    ml.Execute "for i = 1:lgth Reqnom1(i) = Reqnom(i);end;"
    ml.GetWorkspaceData "Reqnom1", "base", temp
    Reqnom.setArray temp
    ml.GetWorkspaceData "lgth", "base", lgth
    ReDim tpsReqnom(0)
    tpsReqnom(0) = Reqnom(0,0)
    for j = 1 to (lgth-1)
    ReDim preserve tpsReqnom(ubound(tpsReqnom) + 1)
    tpsReqnom(j) = Reqnom(0,j)
    next
    Reqnom.setArray tpsReqnom

    ml.Execute "lgth = length(DV_fan)"
    ml.Execute "for i = 1:lgth DV_fan1(i) = DV_fan(i);end;"
    ml.GetWorkspaceData "DV_fan1", "base", temp
    DV_fan.setArray temp
    ml.GetWorkspaceData "lgth", "base", lgth
    ReDim tpsDV_fan(0)
    tpsDV_fan(0) = DV_fan(0,0)
    for j = 1 to (lgth-1)
    ReDim preserve tpsDV_fan(ubound(tpsDV_fan) + 1)
    tpsDV_fan(j) = DV_fan(0,j)
    next
    DV_fan.setArray tpsDV_fan
end sub
```

Appendix I

Architecture Model Builder – Code

This appendix presents the code implementing the the architecture model builder. This code was written in the Java language in the Eclipse environment. This code is structured around seven classes. The code used to implement each class is presented in this appendix.

I.1 BoundaryFunctionBuilder

```
package edu.gatech.models.integration.mc;

import java.util.ArrayList;

import javax.swing.JOptionPane;

import com.phoenix_int.ModelCenter.Array;
import com.phoenix_int.ModelCenter.ModelCenter;
import com.phoenix_int.ModelCenter.ModelCenterException;
import com.phoenix_int.ModelCenter.Variable;

public class BoundaryFunctionBuilder {

	public static boolean ExportBF(ArrayList <String[]> scenariosSchedule, String filePathIn,
			String filePathOut, ModelCenter mc ) throws ModelCenterException {
		// This model sends to the 'Mission' analysis all the information about the scenario so that it can retrieve
		// the proper value for the boundary requirements


		//MODEL CENTER
		//Loads the MC file to modify
		mc.loadFile(filePathIn);
		//ArrayList <String[]> scenariosSchedule=new ArrayList <String[]> ();
		// The element in the scenariosSchedule list a series of sequences. The information contained in the String[] define:
		// [0]- Name of the scenario
		// [1]- The configuration used for the scenario
		// [2]- The criticality level used to defined the requirements
		// [3]- The mission phase
		// [4]- Reinitiation variable (0 if normal scenario, 1 if placeholder for restart)
		int nTotStates = scenariosSchedule.size();
		String Root="Model.Mission.";
		//MODEL CENTER
		SizeVariable(Root+"Scenario",nTotStates,mc);
		SizeVariable(Root+"FlightPhase",nTotStates,mc);
		SizeVariable(Root+"Criticality",nTotStates,mc);
		for (int k = 0; k<nTotStates; k++){
			String[] scenario=scenariosSchedule.get(k);
			mc.setValue(Root+"Scenario["+k+"]", scenario[0]);		// [0]- Name of the scenario
			mc.setValue(Root+"FlightPhase["+k+"]", scenario[3]);		// [1]- The configuration used for the scenario
			mc.setValue(Root+"Criticality["+k+"]", scenario[2]);		// [2]- The criticality level used to defined the requirements
```

```
                }



                // Saving modified MC model
                //MODEL CENTER
                mc.saveModelAs(filePathOut);

                return true;
        }


        public static boolean SizeVariable(String variable, int nTotStates, ModelCenter mc
) throws ModelCenterException {
                //MODEL CENTER
                Variable var=mc.getVariable(variable);
                if (var instanceof Array){
                        Array array = (Array) var;
                        array.setDimensions(nTotStates);
                }else{
                        JOptionPane.showMessageDialog(null, "Error:  "+variable+" is not an
Array.");
                        mc.release();
                        return false;
                }
                return true;
        }
}
```

I.2 CompositionMaker

```
package edu.gatech.models.integration.mc;

import java.awt.event.ActionEvent;
import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;
import java.util.Iterator;

import javax.swing.JOptionPane;

import com.nomagic.magicdraw.openapi.uml.SessionManager;
import com.nomagic.magicdraw.ui.browser.Node;
import com.nomagic.magicdraw.ui.browser.Tree;
import com.nomagic.magicdraw.ui.browser.actions.DefaultBrowserAction;
import com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class;
import com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element;
import com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Property;
import com.nomagic.uml2.ext.magicdraw.compositestructures.mdports.Port;
import com.phoenix_int.ModelCenter.Array;
import com.phoenix_int.ModelCenter.ModelCenter;
import com.phoenix_int.ModelCenter.ModelCenterException;
import com.phoenix_int.ModelCenter.Variable;

public class CompositionMaker extends DefaultBrowserAction{

        private static final long serialVersionUID = 1L;

        public CompositionMaker()
        {
                super("", "Test composition", null, null);
        }

        public void actionPerformed(ActionEvent e)
        {
                if (!SessionManager.getInstance().isSessionCreated())
                {
                        SessionManager.getInstance().createSession("Integration Action");
                }
                Tree tree = getTree();
                for (Integer i = 0; i < tree.getSelectedNodes().length; i++){      // This
loop scans all nodes selected in the MD browser tree
                        Node node = tree.getSelectedNodes()[i];
                        Object userObject = node.getUserObject();
                        if (userObject instanceof Class){    //Action  will  be  applied  to
elements of type "Class" (which includes SysML blocks)
                                Class architectureBlock = (Class) userObject;
                                String        filePathIn="C:\\Program        Files\\Phoenix
```

```
Integration\\Analysis Server 5.1\\analyses\\Archsetup\\MC Start
files\\"+JOptionPane.showInputDialog(null,"Enter file name (without
extention):","Start1")+".pxc";
                        String
filePathOut="C:\\OfficeFolder\\ModelGeneration\\TestComposition.pxc";
                        int totStates=30;
                        CompositionBuilder(architectureBlock,totStates,
filePathIn,filePathOut,true, true);

                    }else
                    JOptionPane.showMessageDialog(null,"Error: Hey genious! This is not
a block!");}
            }
             SessionManager.getInstance().closeSession();
       }

       public static boolean CompositionBuilder(Class architectureBlock, int
totalNstates, String filePathIn,
                    String filePathOut,  boolean showAll,boolean debug){
            // This returns true if all was OK

            //MODEL CENTER
            try
            {
                    ModelCenter mc = new ModelCenter();
                    mc.loadFile(filePathIn);//Loads the MC file to modify
                    //*/
                    Collection <Property> parts= architectureBlock.getPart();
                    Integer nElements = parts.size();
                    if (debug){JOptionPane.showMessageDialog(null, "Message 01 -
 "+"There are "+nElements+" elements in this architecture");}// FOR TESTING
                    Integer Counter =0; String Message="";String ListOfVarTargets="";
                    // Recording MC variable name path within subsystem model
                    HashMap <String,String> varPathForSubsystem = new HashMap
<String,String>();


                    // Registering attributes
                    ArrayList <String> attList = new ArrayList<String>();// Lists
attribute types (weight, drag, cost, ect...)
                    HashMap <String,String> attFormats = new
HashMap<String,String>();//  Format of the attribute (double, doublePerState,...)
                    HashMap <String,ArrayList<String>> attPartLists = new
HashMap<String,ArrayList<String>>();//  Lists of subsystems generating values with this
given attribute
                    HashMap <String,ArrayList<String[]>> attFormulas = new HashMap
<String,ArrayList<String[]>>();// List formula set associated with each attribute type


                    // Registering optimization factors
                    ArrayList <Integer> gammaSize = new ArrayList<Integer>();// Keeps
track of the size of the gamma vectors produced by optimizer
                    ArrayList <String[]> gammaFormulas = new ArrayList <String[]>();//
Register gamma formulas
                    ArrayList <String> gammaNames = new ArrayList<String>();// List the
MC name of the gamma variables for each optimized subsytem

                    for (Iterator <Property> iter = parts.iterator(); iter.hasNext();
){
                        Property part = iter.next();// This scans the part
composing the architecture
                        Counter = Counter +1;
                        String partName = part.getName();// Name of the subsystem
in the architecture: e.g. eng1
                        String partType = part.getType().getName();// Type of the
subsystem: e.g. Turbofan engine
                        Collection<Element> partElements =
part.getType().getOwnedElement();// Grabs all elements defining the subsystems considered


                        // Import analysis in the MC model
                        ////////////////////////////////////

                        String modelName ="aserv://localhost/Archsetup/Builder
Library/" + partType;// This sets the model name by default
                        String varNamePath ="";

                        //This verifies that no special model name was used
                        for (Iterator <Element> elIter = partElements.iterator();
elIter.hasNext(); ){
```

```
                                    Element thisElement = elIter.next();//This scans the
elements in the part

                                    if   (thisElement   instanceof   Property){//If   the
element is a property
                                        Property      valueProp      =      (Property)
thisElement;//Cast the proper class name...
                                        String                                valueType=
valueProp.getType().getName();//... and get its type.
                                        if
(valueType.equals("MCmodelReference")){//If   the   type   is   marked   as   an   MC   model
reference...
                                            String
path=valueProp.getDefault();//...get the model path
                                            String           nameInAS           =
valueProp.getName();//...get the model name in the Analysis server
                                            modelName=path+nameInAS;
                                        }
                                    }
                                }
                                mc.createComponent(modelName, partName, "Model");
                                varPathForSubsystem.put(partType,varNamePath);



                                //For  the  element  just  imported  this  will  explore  the
elements of this part for ports, attributes and optimization configuration
                                for  (Iterator  <Element>  elIter  =  partElements.iterator();
elIter.hasNext(); ){
                                    Element thisElement = elIter.next();//This scans the
elements in the part

                                    if   (thisElement   instanceof   Property){//If   the
element is a (value) property
                                        Property      valueProp      =      (Property)
thisElement;//Cast the proper class name...
                                        String                                valueType=
valueProp.getType().getName();//... and get its type.


                                        // Detection of synthesis attributes
                                        ////////////////////////////////////
                                        //ArrayList    <String>    attList    =    new
ArrayList<String>();// Lists attribute types (weight, drag, cost, ect...)
                                        //HashMap  <String,String>  attFormat  =  new
HashMap<String,String>();//  Format of the attribute (double, doublePerState,...)
                                        //HashMap          <String,ArrayList<String>>
attPartLists = new HashMap<String,ArrayList<String>>();//  Lists of subsystems generating
values with this given attribute
                                        //HashMap        <String,ArrayList<String[]>>
attFormulas = new HashMap <String,ArrayList<String[]>>();// List formula set associated
with each attribute type

                                        if
(valueType.startsWith("SubsystemAttribute")){//If the type is marked as an attribute...
                                            String
attType=valueProp.getName();//...get its name (on the synthesis model side)...
                                            String           varPath           =
varPathForSubsystem.get(partType);//Retrieve   the   variable   path   for   the   subsystem
considered

                                            // Define the format of the attribute
                                            String attFormat = "double";
                                            if
(valueType.equals("SubsystemAttributeVector")){attFormat="doublePerState";
}
                                            String
variableName=valueProp.getDefault();//...retrieve its name (on the subsystem model side)
                                            Integer              rankVar=0;Integer
subsystemRank=0;
                                            ArrayList<String[]>      formulas=new
ArrayList<String[]>();//Initialize the formula array and counter
                                            ArrayList<String>     attPartList=new
ArrayList<String>();//Initialize the list with subsystems emiting this attribute
                                            // Checks if this attribute has been
seen before
                                            if(attList.contains(attType)){//It
came up before

        formulas=attFormulas.get(attType);//Get previous formulas
```

```
        attPartList=attPartLists.get(attType);//Get the list of parts

        rankVar=formulas.size();//Determine how many time this attribute has been detected
before
                                                        }else{// if it's the first time this
attribute type is encountered

        attList.add(attType);//Register this type of attribute
                                                                attFormats.put(attType,
attFormat);//Register the format of the variable
                                                        }
                                                        subsystemRank                       =
attPartList.size();//Get the number of times this attribute was observed before
                                                        String  dispAtt="";//  Display  string
to show the type of attribute and its associated formulas
                                                        String   formString="";   //   Display
string to show the formula list
                                                        if (attFormat.equals("double")){// If
the format is double (i.e. 1 variable per subsystem)
                                                                dispAtt="Format:
"+attFormat+"(1)\nAttribute detected: "+attType+
                                                                "\nRank      of      subsystem:
"+subsystemRank;
                                                                String
formula="Model."+partName+varPath+"."+variableName;//Define  the  subsystem  side  variable
MC name
                                                                String         target         =
"Model.SystemSynthesis."+attType+"["+rankVar+"]";//Define the synthesis side variable MC
name
                                                                String[]
fullFormula={target,formula
};

        formulas.add(fullFormula);attFormulas.put(attType,   formulas);//   Register   the
formula
                                                                attPartList.add(partName);//
Add the part name to the list of elements generating this attribute
                                                                attPartLists.put(attType,
attPartList);
                                                                formString="\n"+target+"     =
"+formula;
                                                                dispAtt=dispAtt+formString;
                                                        }else
if(attFormat.equals("doublePerState")){
                                                                dispAtt="Format:
"+attFormat+"(2)\nAttribute detected: "+attType+
                                                                "\nRank      of      subsystem:
"+subsystemRank;
                                                                String
formulaRoot="Model."+partName+varPath+"."+variableName;//Define    the    subsystem    side
variable MC name
                                                                String       targetRoot       =
"Model.SystemSynthesis."+attType;//Define the synthesis side variable MC name
                                                                for                          (int
state=0;state<totalNstates;state++){
                                                                        String    formula    =
formulaRoot+"["+state+"]";
                                                                        String     target     =
targetRoot+"["+subsystemRank+","+state+"]";
                                                                        String[]
fullFormula={target,formula
};// Format: SynthesisVar[rank,state]=SubsystemVar[state]

        formulas.add(fullFormula);attFormulas.put(attType,   formulas);//   Register   the
formula

        formString=formString+"\n"+target+" = "+formula;

        dispAtt=dispAtt+formString;
                                                                }
                                                                attPartList.add(partName);//
Add the part name to the list of elements generating this attribute
                                                                attPartLists.put(attType,
attPartList);
                                                        }
                        if(showAll){
                                JOptionPane.showMessageDialog(null,dispAtt);}

                                                }
```

```
                                                //Note: The MC implement is performed after
all components in the architecture are detected. We need...
                                                //... to pre-size the variable before links
are created. Therefore this MC implementation will occur...
                                                //... once all parts were explored (i.e.
once we know how many time the attribute is taken to the ...
                                                //... synthesis analysis).


                                                // Detection of optimization parameters
                                                ////////////////////////////////////////

                                                if
(valueType.equals("OptimizationParameter")){//If the type is marked as an optimization
parameter...
                                                        String
gammaName=valueProp.getName();//...get its name (on the subsystem model side)...

                                                        String
ngammaS=valueProp.getDefault();//...and the number of gamma in the optimization
                                                        Integer ngamma= new Integer(ngammaS);
//parseInt(ngammaS);
                                                        int rank=gammaSize.size();
                                                        String formString="";
                                                        gammaSize.add(ngamma);//Let       the
optimizer know how many gamma he is expected to optimize
                                                        for (int n=0;n<ngamma;n++){
                                                                String        varPath        =
varPathForSubsystem.get(partType);
                                                                String
gammaMCName="Model."+partName+varPath+"."+gammaName;
                                                                gammaNames.add(gammaMCName);
                                                                String                    target
=gammaMCName+"["+n+"]";//Define the subsystem side variable MC name
                                                                String
formula="Model.Optimization.gammas["+rank+","+n+"]";//Define the optimizer side variable
MC name
                                                                String[]
fullFormula={target,formula
};

        gammaFormulas.add(fullFormula);// Register the formula

        formString=formString+"\n"+target+" = "+formula;
                                                        }// Done iterating on each gamma
                                                }//
                                        }


                                        // Initialization of state dependent arrays
                                        ///////////////////////////////////////////

                                        if (thisElement instanceof Port){// If the element
is a port
                                                Port port = (Port) thisElement;
                                                String  portName  =  port.getName();//Get  its
name...
                                                Class          functionBlock=          (Class)
port.getType();//... and function type
                                                String fctType=functionBlock.getName();
                                                String portIndex = portName.substring(3);
                                                if  (portName.startsWith("Cap")){//  Check  if
it is a capability or req
                                                        int[] relevantTypes={1,3,4
};// 2 types of functional relationship will imply
                                                        //state-dependent    inputs    for    a
capability port:
                                                        // - requirement flow (type=1)
                                                        //    -    specification    flow    to
source(type=3)
                                                        //  -  specification  flow  to  load
(type=34)
                                                        for (int ty=0;ty<3;ty++){
                                                                int type=relevantTypes[ty];
                                                                if
(ConfStructureHelper.FctFlowExistance(fctType, functionBlock, type)){// checks if
                                                                                        //a flow is implied by
its function
                                                                                        //     Import     name
```

```
conventions from the function block
                                                                ArrayList   <String[]>
nameConvention = ConfStructureHelper.MCVarNameConventions(fctType, functionBlock, type);
                                                                int     varSize     =
nameConvention.size(); //Determine the # of variables implied by the flow
                                                                for                 (int
varRank=0;varRank<varSize;varRank++){// For each  variable...
                                                                String[]  vName
= nameConvention.get(varRank);// ...get its name convention...
                                                                String
target="";//... and build its MC name
                                                                String  varPath
= varPathForSubsystem.get(partType);
                                                                if
(portIndex.length()>0){ // The system provides more than 1 capability
        target="Model."+partName+varPath+"."+vName[1]+portIndex;// Following VarFormat 1
                                                                }else{
        target="Model."+partName+varPath+"."+vName[1];// Following VarFormat 1
                                                                }
        ListOfVarTargets=ListOfVarTargets+"\n"+target;//For testing


                                                                //Formating
array in MC
                                                                //MODEL CENTER
                                                                Variable
var=mc.getVariable(target);
                                                                if         (var
instanceof Array){
                                                                        Array
array = (Array) var;
        array.setDimensions(totalNstates);
                                                                }else{
        JOptionPane.showMessageDialog(null, "Error:  "+target+" is not an Array.");
        mc.release();return false;
                                                                }//*/
                                                            }//Done    with    all
variables in the flow
                                                        }// Done with flow
                                                    }// Done with type iteration
                                                }else if(portName.startsWith("Req")){
                                                    if (partType.equals("BoundaryFct")){
                                                    }else{
                                                        int[] relevantTypes={2,4
};// 2 types of functional relationship will imply
                                                        //state-dependent  inputs  for
a capability port:
                                                        //   -   characteristic   flow
(type=2)
                                                        //  -  specification  flow  to
requirement (type=4)
                                                        for (int ty=0;ty<2;ty++){
                                                            int
type=relevantTypes[ty];
                                                            if
(ConfStructureHelper.FctFlowExistance(fctType, functionBlock, type)){// checks if
                                                            //a   flow   is
implied by its function

                                                            // Import name
conventions from the function block
                                                            ArrayList
<String[]>     nameConvention     =     ConfStructureHelper.MCVarNameConventions(fctType,
functionBlock, type);
                                                            int  varSize  =
nameConvention.size(); //Determine the # of variables implied by the flow
                                                            for         (int
varRank=0;varRank<varSize;varRank++){// For each  variable...
        String[] vName = nameConvention.get(varRank);// ...get its name convention...
                                                                        String
target="";//... and build its MC name
                                                                        String
varPath = varPathForSubsystem.get(partType);
```

```
                                                                                if
(portIndex.length()>0){ // The system provides more than 1 capability
        target="Model."+partName+varPath+"."+vName[2]+portIndex;// Following VarFormat 2
                                                                                }else{
        target="Model."+partName+varPath+"."+vName[2];// Following VarFormat 2
                                                                                }
        ListOfVarTargets=ListOfVarTargets+"\n"+target;//For testing


        //Formating array in MC
                                                                                //MODEL
CENTER
        Variable var=mc.getVariable(target);
                                                                                if (var
instanceof Array){
        Array array = (Array) var;
        array.setDimensions(totalNstates);
                                                                                }else{
        JOptionPane.showMessageDialog(null, "Error:  "+target+" is not an Array.");
        mc.release();return false;
                                                                                }//*/
                                                                            }//Done    with
all variables in the flow
                                                                        }// Done with flow
                                                                    }// Done with type iteration
                                                                }// Done if req port
                                                            }else{
        JOptionPane.showMessageDialog(null,"Error in Compositionbuilder: \nSome ports are
neither Cap nor Req");// Displays the list
                                                                mc.release();return true;
                                                            }//Done with port type
                                                        }//Done with port
                                                    }// Done scanning elements in part
                                                    Message = Message + Counter + "- partName: " + partName + "
of type : " + partType + "\n"; //Lists the part names and their type


                                                    // Scenario links between mission and the subsystems
                                                    ////////////////////////////////////////////////////
        mc.createLink("Model."+partName+".DT","Model.Mission.DT");//Length  of  the  time
step
        mc.createLink("Model."+partName+".Cruisek","Model.Mission.Cruisek");//Pointer   on
cruise phase



                                }//Done scanning all parts in the architecture


                                //Synthesis attributes in MC
                                ////////////////////////////
                                //ArrayList  <String>  attList  =  new  ArrayList<String>();//  Lists
attribute types (weight, drag, cost, ect...)
                                //HashMap       <String,String>       attFormat       =       new
HashMap<String,String>();//  Format of the attribute (double, doublePerState,...)
                                //HashMap    <String,ArrayList<String>>    attPartLists    =    new
HashMap<String,ArrayList<String>>();//  Lists of subsystems generating values with this
given attribute
                                //HashMap  <String,ArrayList<String[]>>  attFormulas  =  new  HashMap
<String,ArrayList<String[]>>();// List formula set associated with each attribute type


                                int  nAtt=attList.size();String  DisplayAttResize="Review  of  the
resized attribute synthesis variables:";
                                //presizing variable array
                                for  (int  attRank=0;attRank<nAtt;attRank++){//  Iterates  on  each
attribute category
                                        String attType= attList.get(attRank);
```

```
                                String attFormat = attFormats.get(attType);//Retrieve the
format of the variable
                                ArrayList<String> attPartList = attPartLists.get(attType);
                                Integer nSubsystems = attPartList.size();//Number of
subsystems emiting this variables
                                ArrayList<String[]> formulas= attFormulas.get(attType);
                                int sizeAttArray=formulas.size();//determine the number of
input to this array
                                String MCtarget = "Model.SystemSynthesis."+attType;//(re-
)define its MC name
                                Variable var=mc.getVariable(MCtarget);
                                if (var instanceof Array){
                                        Array array = (Array) var;
                                        if (attFormat.equals("double")){
                                                array.setDimensions(sizeAttArray);

        DisplayAttResize=DisplayAttResize+"\n"+MCtarget+":
["+nSubsystems+","+totalNstates+"]";
                                        }
                                        else if(attFormat.equals("doublePerState")){

        array.setDimensions(nSubsystems,totalNstates);

        DisplayAttResize=DisplayAttResize+"\n"+MCtarget+":
["+nSubsystems+","+totalNstates+"]";
                                        }
                                        // Export links bringing subsystem attributes to
system synthesis
                                        for (int item=0;item<sizeAttArray;item++){
                                                String[] fullFormula=formulas.get(item);
                                                String target=fullFormula[0];
                                                String formula=fullFormula[1];
                                                mc.createLink(target, formula);
                                        }
                                }else{
                                        JOptionPane.showMessageDialog(null,          "Error:
"+MCtarget+" is not an Array.");
                                        mc.release();
                                        return false;
                                }

                        }


                        //Optimization parameters in MC
                        ////////////////////////////////
                        //Definition of the gammaSize variable (Variable telling the
optimizer how many gammas are required)
                        int optSet= gammaSize.size();// Detect the number of subsystems
requiring gamma optimization
                        String MCtarget = "Model.Optimization.gammaSize";//define its MC
name
                        //Step 1: sizing gammaSize
                        Variable var=mc.getVariable(MCtarget);
                        if (var instanceof Array){
                                Array array = (Array) var;
                                array.setDimensions(optSet);

                                for (int n=0;n<optSet;n++){//Iterate optimized subsystem by
optimized subsystem
                                        //Step 3.1: Definition of gammaSize values
                                        String Target=MCtarget+"["+n+"]";
                                        String ngamma=gammaSize.get(n).toString();
                                        //MODEL CENTER
                                        mc.setValue(Target,ngamma);

                                        //Step 3.2: Definition of gamma size for each
subsystem
                                        String   gammaMCName=gammaNames.get(n);//gammaMCName
is not returning the correct name value
                                        Variable gammaVar=mc.getVariable(gammaMCName);
                                        if (gammaVar instanceof Array){
                                                //Array gammaArray = (Array) gammaVar;
                                                //TODO: Temporary fix. The pre-sizing of
gamma is not necessary and was incorrect. See the problem 4 lines above
                                                //Integer ngammaInt= new Integer(ngamma);
                                                //JOptionPane.showMessageDialog(null,
"gammaMCName:  "+gammaMCName+"\nngamma: "+ngamma);
                                                //gammaArray.setDimensions(ngammaInt);
                                        }else{
```

```
                                        JOptionPane.showMessageDialog(null,   "Error:
"+gammaMCName+" is not an Array.");
                                        mc.release();
                                        return false;
                                }
                        }//Done iterating over each optimized subsystem
                        for                                                     (int
nform=0;nform<gammaFormulas.size();nform++){//Building the links
                                String[] fullFormula=gammaFormulas.get(nform);
                                String target=fullFormula[0];
                                String formula=fullFormula[1];
                                mc.createLink(target, formula);
                        }
                }else{
                        JOptionPane.showMessageDialog(null,  "Error:    "+MCtarget+"
is not an Array.");
                        mc.release();
                        return false;
                }

                if (showAll){
                        JOptionPane.showMessageDialog(null,"Message       02       -
"+Message);// Displays the list of architecture elements
                        JOptionPane.showMessageDialog(null,"Message       03       -
"+ListOfVarTargets);// Displays the list of state dependent arrays
                }

                // Saving modified MC model
                //MODEL CENTER
                mc.saveModelAs(filePathOut);
                mc.release();
                return true;
        }catch ( ModelCenterException mcException ){
                    JOptionPane.showMessageDialog(null, mcException.getMessage());
                    assert false;
                    return false;
        }//*/
    }




    public void updateState(){
        if (getTree().getSelectedNodes().length > 1)
            this.setEnabled(false);
    }
}
```

I.3 ConfStructureHelper

```
package edu.gatech.models.integration.mc;

import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;
import java.util.Iterator;

import javax.swing.JOptionPane;

import
com.nomagic.uml2.ext.magicdraw.auxiliaryconstructs.mdinformationflows.InformationFlow;
import com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class;
import com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Classifier;
import com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element;
import com.nomagic.uml2.ext.magicdraw.classes.mdkernel.NamedElement;
import com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Property;
import com.nomagic.uml2.ext.magicdraw.compositestructures.mdinternalstructures.Connector;
import
com.nomagic.uml2.ext.magicdraw.compositestructures.mdinternalstructures.ConnectorEnd;
import com.phoenix_int.ModelCenter.ModelCenter;
import com.phoenix_int.ModelCenter.ModelCenterException;

public class ConfStructureHelper {

    public static boolean ConfStructureBuilder(Class confBlock,Integer StateRk,String
filePathIn,
```

```
                        String filePathOut,boolean debug, boolean showAll, ModelCenter mc )
throws ModelCenterException {

                //MODEL CENTER
                //Loads the MC file to modify
                mc.loadFile(filePathIn);

                /* Step 1: Import the connection from the configuration
                ------------------------------------------------------*/
                // This is a table which list connection by specifying the Cap and Ref...
                // This scans each element contained in the class (line 68)
                ArrayList <String[]> TableCon = new ArrayList <String[]>();
                HashMap <String,Class> FunctionBlocks= new HashMap <String,Class>();
                HashMap     <String,String>     varPathForSubsystem     =     new     HashMap
<String,String>();
                ArrayList<String[]> varPathArray = new ArrayList<String[]>();
                String printMe=""; Integer Counter=1;// FOR TESTING
                // Step 1.1
                for (Iterator <NamedElement> iter = confBlock.getOwnedMember().iterator();
iter.hasNext(); ){
                        NamedElement namedElem = iter.next();
                        // Step 1.2: Until it gets to a connector
                        if (namedElem instanceof Connector){
                                Connector   conn   =   (Connector)   namedElem;   //   Stores
connector object in conn

                                Iterator         <ConnectorEnd>         connEnds         =
conn.getEnd().iterator(); // The ends of the connector is stored in connEnds
                                ConnectorEnd connEnd1 = connEnds.next();
                                ConnectorEnd connEnd2 = connEnds.next();
                                Property partProperty1 = connEnd1.getPartWithPort();
                                Property partProperty2 = connEnd2.getPartWithPort();
                                int counter=0;

                                // Step 1.3
                                String reqSID = null;String reqSType = null; String reqPID
= null;
                                String capSID = null;String capSType = null;String capPID
= null;
                                String  functionType  =  null;  String  reqType=null;String
capType=null;
                                // This condition identify the sourceEnd and the loadEnd
of the connection
                                if (connEnd2.getRole().getName().startsWith("Cap")){
                                        // Import the identify of the capability port
                                        capSID                                          =
connEnd2.getPartWithPort().getName();//Name of the subsystem implementing the requirement
                                        capSType = partProperty2.getType().getName();//Type
of the subsystem implementing the requirement
                                        capPID = connEnd2.getRole().getName();//Name of the
functional port on the system implementing the requirement
                                        capType                                         =
connEnd2.getRole().getType().getName();//Type  of  the  functional  port  on  the  system
formulating the requirement

varPathArray=DefineVarArray(partProperty2,varPathForSubsystem,varPathArray);

varPathForSubsystem=DefineVarPath(partProperty2,varPathForSubsystem);
                                        if
(showAll){counter++;JOptionPane.showMessageDialog(null,   "ID   card   for   port:\ncapSID:
"+capSID+"\ncapSType: "+capSType+

                                                                "\ncapPID:     "+capPID+"\nID     of
port:"+capPID.substring(3)+"\ncapPType: "+capType);} // FOR TESTING

                                        // Determine if requirement is a BF or IF
                                        if
(connEnd2.getRole().getName().startsWith("Req")){// This is an induced requirements
                                                reqSID                                  =
connEnd1.getPartWithPort().getName();//Name of the subsystem formulating the requirement
                                                reqSType                                =
partProperty1.getType().getName();//Type of the subsystem formulating the requirement
                                                reqPID                                  =
connEnd1.getRole().getName();//Name of the functional port on the system formulating the
requirement
                                                reqType                                 =
connEnd1.getRole().getType().getName();//Name  of  the  functional  port  on  the  system
formulating the requirement

varPathArray=DefineVarArray(partProperty1,varPathForSubsystem,varPathArray);//  Import  the
```

```
variable namepath corresponding to the subsystem type

varPathForSubsystem=DefineVarPath(partProperty1,varPathForSubsystem);
                                        if
(showAll){JOptionPane.showMessageDialog(null,     "ID     card     for     port:\nreqSID:
"+reqSID+"\nreqSType: "+reqSType+
                                                "\nreqPID: "+reqPID+"\nID of
port:"+reqPID.substring(3)+"\nreqPType: "+reqType);} // FOR TESTING
                                    }else{ // This is a boundary function
                                        reqSID   =   "BoundaryFct";//Name   of   the
subsystem formulating the requirement
                                        reqSType                                    =
partProperty1.getType().getName();//Type of the subsystem formulating the requirement
                                        reqPID = connEnd1.getRole().getName();//Name
of the functional port on the system formulating the requirement
                                        reqType                                     =
connEnd1.getRole().getType().getName();//Name  of  the  functional  port  on  the  system
formulating the requirement
                                        if
(showAll){JOptionPane.showMessageDialog(null,     "ID     card     for     port:\nreqSID:
"+reqSID+"\nreqSType: "+reqSType+
                                                "\nreqPID: "+reqPID+"\nID of
port:"+reqPID.substring(3)+"\nreqPType: "+reqType);} // FOR TESTING
                                    }
                                }                           else                            if
(connEnd1.getRole().getName().startsWith("Cap")){
                                        capSID    =    connEnd1.getPartWithPort().getName();
       //Name of the subsystem implementing the requirement
                                        capSType = partProperty1.getType().getName();//Type
of the subsystem implementing the requirement
                                        capPID = connEnd1.getRole().getName();//Name of the
functional port on the system formulating the requirement
                                        capType                                     =
connEnd1.getRole().getType().getName();//Type  of  the  functional  port  on  the  system
formulating the requirement

varPathArray=DefineVarArray(partProperty1,varPathForSubsystem,varPathArray);// Import the
variable namepath corresponding to the subsystem type

varPathForSubsystem=DefineVarPath(partProperty1,varPathForSubsystem);
                                        if
(showAll){counter++;JOptionPane.showMessageDialog(null,   "ID   card   for   port:\ncapSID:
"+capSID+"\ncapSType: "+capSType+
                                                    "\ncapPID:     "+capPID+"\nID     of
port:"+capPID.substring(3)+"\ncapPType: "+capType);} // FOR TESTING
                                    // Determine if requirement is a BF or IF
                                        if
(connEnd2.getRole().getName().startsWith("Req")){// This is an induced requirements
                                        reqSID                                      =
connEnd2.getPartWithPort().getName();//Name of the subsystem formulating the requirement
                                        reqSType                                    =
partProperty2.getType().getName();//Type of the subsystem formulating the requirement
                                        reqPID                                      =
connEnd2.getRole().getName();//Name of the functional port on the system formulating the
requirement
                                        reqType                                     =
connEnd2.getRole().getType().getName();//Name  of  the  functional  port  on  the  system
formulating the requirement

varPathArray=DefineVarArray(partProperty2,varPathForSubsystem,varPathArray);// Import the
variable namepath corresponding to the subsystem type

varPathForSubsystem=DefineVarPath(partProperty2,varPathForSubsystem);
                                        if
(showAll){JOptionPane.showMessageDialog(null,     "ID     card     for     port:\nreqSID:
"+reqSID+"\nreqSType: "+reqSType+
                                                "\nreqPID: "+reqPID+"\nID of
port:"+reqPID.substring(3)+"\nreqPType: "+reqType);} // FOR TESTING
                                    }else{ // This is a boundary function
                                        reqSType = "BoundaryFct";//Marking that this
is a boundary function
                                        reqSID = connEnd2.getRole().getName();//Name
of the functional port on the system formulating the requirement
                                        reqPID = "Req";
                                        reqType                                     =
connEnd2.getRole().getType().getName();//Name  of  the  functional  port  on  the  system
formulating the requirement
                                        if
(showAll){JOptionPane.showMessageDialog(null, "ID card for port:\nreqSType: "+reqSType+
                                                "\nreqPID:
```

```
"+reqPID+"\nreqPType: "+reqType);} // FOR TESTING
                                        }
                                }
                                // Step 1.6: This condition registers the function block
so that its internal variable conventions can be accessed when links are defined
                                Class fctBlock = (Class) connEnd1.getRole().getType();
                                String fctName=connEnd1.getRole().getType().getName();
                                if (FunctionBlocks.containsKey(fctName)){
                                }else{
                                        FunctionBlocks.put(fctName, fctBlock);
                                }
                                fctBlock = (Class) connEnd2.getRole().getType();
                                fctName=connEnd2.getRole().getType().getName();
                                if (FunctionBlocks.containsKey(fctName)){
                                }else{
                                        FunctionBlocks.put(fctName, fctBlock);
                                }
                                // Check the type of function implied by the connector
                                Integer
nfct=conn.get_informationFlowOfRealizingConnector().size();// Number of functions on the
connection

                                //Step 1.4
                                if
(conn.get_informationFlowOfRealizingConnector().size()>1){
                                        String ErrorMessage="Error: More than one function
is conveyed here. Check you diagram genius!!!\n";
                                        ErrorMessage=ErrorMessage+"Check connection between
"+reqSID+" and "+capSID;
                                        JOptionPane.showMessageDialog(null, ErrorMessage);
                                        return false;
                                }else{
                                        if   (nfct.equals(0)){//No   functional   flow   was
assigned
                                                functionType=capType;
                                                String  ErrorMessage="Warning:  No  function
flow was assigned between "+reqSID+" and "+capSID;
                                                ErrorMessage=ErrorMessage+"\n.    The    flow
assumed was "+functionType;
                                                if
(debug){JOptionPane.showMessageDialog(null, ErrorMessage);
}
                                        }else{//One functional flow was assigned
                                                InformationFlow      informationFlow      =
conn.get_informationFlowOfRealizingConnector().iterator().next();
                                                Classifier            conveyed            =
informationFlow.getConveyed().iterator().next();
                                                functionType = conveyed.getName();//Type of
the function conveyed by the relationship
                                                if
(FunctionBlocks.containsKey(functionType)){//  verifies  if  this  function  type  was  not
registered (step 1.6)
                                                }else{
                                                        Class    functionBlock    =    (Class)
conveyed;
                                                        FunctionBlocks.put(functionType,
functionBlock);
                                                }
                                        }
                                        //  Verify  that  ports  and  functional  flow  are
compatible
                                        if
(FunctionalHandshake(reqPID,capPID,reqType,capType,functionType)){
                                                //Step    1.5:    Stores    the    information
characterizing the new connection in the list
                                                String[]                IDs                =
{capSID,reqSID,capSType,reqSType,capPID,reqPID,functionType
};
                                                /* Based on this definition each connector
is defined as:
                                                        [0]: CapSID (name of subsystem)
        [1]: LoadID or ReqSID
                                                        [2]: capSType (type of subsystem)
        [3]: reqType
                                                        [4]: PortSourceID or CapPID
        [5]: reqPID
                                                        [6]: FunctionType*/
                                                TableCon.add(IDs);
                                                printMe  =  printMe  +  Counter  +  "     "  +
capSID + "("+capSType+")>"+functionType+">" + reqSID + "("+reqSType+")\n";// FOR TESTING
```

```
                                        Counter++;// FOR TESTING

                                    }else{
                                        String ErrorMessage="Definition error: The
ports and function flow do not agree.\n";
                                        ErrorMessage=ErrorMessage+"Check the
connection between "+reqSID+" and "+capSID;

JOptionPane.showMessageDialog(null,ErrorMessage);
                                        return false;
                                    }//End of valid/invalid handshake
                                }//End of connector validity verification
                        }// Done scanning the connector
                }
                if (debug){JOptionPane.showMessageDialog(null, printMe);
} // FOR TESTING

                /* Step 2: Identification of nodes and # of connections to each
                ----------------------------------------------------------------*/
                Integer count =0;//internal variable counting the number of connections to
each node
                Integer Totcon = TableCon.size();    //   Total   #   of   connections   in
configuration
                ArrayList <String[]> TableCap = new ArrayList <String[]>();// Table
listing the Cap nodes with their key caracteristics
                // CapNodes
                HashMap      <String,Integer>      CapNodes      =      new      HashMap
<String,Integer>();//Indexed map including Cap nodes
                for (int conn = 0; conn<Totcon; conn++){
                        String[] IDs=TableCon.get(conn);
                        String  capNodeID  =  IDs[0]+"."+IDs[4];//  Defines  the  string:
CapPortID.CapSubsystemID
                        if (CapNodes.containsKey(capNodeID)){
                                count = CapNodes.get(capNodeID)+1; // Increase the number
of connection from this node
                        } else {
                                count = 1;// Initialize the number of connections from this
new capability node
                                String[] capfullDef={IDs[0],IDs[4],IDs[6],IDs[2]
};//CapSID,CapPID,FctType,CapType
                                TableCap.add(capfullDef);
                        }
                        CapNodes.put(capNodeID,new  Integer(count));//  Create/update  the
number of connection
                }

                // ReqNodes
                HashMap      <String,Integer>      ReqNodes      =      new      HashMap
<String,Integer>();//Indexed map including Req nodes
                ArrayList <String[]> TableReq = new ArrayList <String[]>();// Table
listing the req nodes with their key caracteristics
                for (int conn = 0; conn<Totcon; conn++){
                        String[] IDs=TableCon.get(conn);
                        String  reqNodeID  =  IDs[1]+"."+IDs[5];//  Defines  the  string:
CapPortID.CapSubsystemID
                        if (ReqNodes.containsKey(reqNodeID)){
                                count = ReqNodes.get(reqNodeID)+1; // Increase the number
of connection
                        } else {
                                count = 1;// Initialize the number of connections
                        }
                        ReqNodes.put(reqNodeID,new  Integer(count));//  Create/update  the
number of connection
                        String[] reqfullDef={IDs[1],IDs[5],IDs[6],IDs[3]
};
                        TableReq.add(reqfullDef);
                }


                /*  Step 3: Preparing equations:
                --------------------------------*/

                //Step 3.1: Realization of requirement flows
                int nCap = CapNodes.size();String Message="";
                for (int tarket = 0; tarket<nCap; tarket++){ //This loop scan all the
capacity ports in architecture conf.
                        String[]  capfullDef  =TableCap.get(tarket);  //  Load  capability
identity
                        String CapSID=capfullDef[0]; //  ID  of  the  system  providing
capability
```

```
                        String CapPID=capfullDef[1];  //   ID   of   the   port   providing
capability
                        String Capfct=capfullDef[2];  // function provided by the port
                        String CapType=capfullDef[3]; //   Type   of   the   port   providing
capability
                        Class fctBlock = FunctionBlocks.get(Capfct);

                        //Step 3.1a: Verifies that the port expects a requirement flow
                        boolean fReqExist = FctFlowExistance(Capfct,fctBlock,1);
                        if (fReqExist){
                                ArrayList <String[]> ReqRel = new ArrayList <String[]>();
                                int CounterCon=0;
                                // Step 3.1b: Scan each connections
                                for (int conn=0; conn<TableCon.size(); conn++){
                                        String[] IDs=TableCon.get(conn);
                                        if (IDs[0].equals(CapSID)&&IDs[4].equals(CapPID)){//
If system and port correspond
                                                CounterCon++;// Increment counter for total
# of connections
                                                String ReqSID = IDs[1];       //     Register
system ID of the requirement
                                                String ReqPID = IDs[5];       //     Register
port ID for the requirement
                                                String ReqNodeID= ReqSID+"."+ReqPID;
                                                String                  nRel                =
ReqNodes.get(ReqNodeID).toString(); //Number of systems support the requirement
                                                String        Function=IDs[6];        String
ReqType=IDs[3];
                                                String[]               reqRel               =
{ReqSID,ReqPID,nRel,ReqType,Function
};
                                                /*  The  requirement  relationship  is  defined
by:
                                                [0]: ReqSID                             [1]:
ReqPID
                                                [2]: nRel                               [3]:
ReqType
                                                [4]: Function
                                                */
                                                ReqRel.add(reqRel);
                                }}

                                // This paragraph is only to display the relationships in a
message box
                                printMe = CapSID + "["+CapPID+"] = ";// FOR TESTING
                                String[] reqRel=ReqRel.get(0);
                                for (int rel=0;rel<ReqRel.size();rel++){
                                        reqRel=ReqRel.get(rel);
                                        printMe=printMe+"1/"+reqRel[2]+"                    x
"+reqRel[0]+"["+reqRel[1]+"]"; // FOR TESTING
                                        if (ReqRel.size()>rel+1){printMe=printMe+ " + ";
}// FOR TESTING
                                }
                                Message = Message + printMe+"\n";// FOR TESTING

                                // Step 3.1c: Generating MC variable names
                                ArrayList                                   <String[]>
Formulas=MCreqlinkMaker(CapSID,CapPID,CapType,ReqRel,StateRk,FunctionBlocks,
                                varPathForSubsystem, debug, showAll);

                                // Step 3.1d: Exporting formulas to MC
                                int varSize=Formulas.size();
                                for (int varN=0; varN<varSize;varN++){
                                        String[] Formulaterms=Formulas.get(varN);
                                        //MODEL CENTER
                                        mc.createLink(Formulaterms[0], Formulaterms[1]);
                                        Message=Message+CapSID+":     "+Formulaterms[0]+"="+
Formulaterms[1]+"\n";// FOR TESTING
                                }
                                Message=Message+"\n";
                        }// End of requirement flow condition
                }//End of scanning the capability ports
                if (debug){JOptionPane.showMessageDialog(null, Message);
} // FOR TESTING


                // Step 3.2: Realization of specification flows to capability ports
                ArrayList <String> specDefined=new ArrayList<String>(); /*Some capability
are used multiple times. To unsure
                that we don't redefine the specs multiple times, this array keeps track of
```

```
them*/

              nCap = CapNodes.size(); Message="Specification flows to capability Ports:
\n";
              for (int tarket = 0; tarket<nCap; tarket++){ //This loop scan all the
capacity ports in architecture conf.
                     String[]  capfullDef  =TableCap.get(tarket);  //  Load  capability
identity
                     String CapSID=capfullDef[0];  //   ID   of   the   system   providing
capability
                     String CapPID=capfullDef[1];  //   ID   of   the   port   providing
capability
                     String Capfct=capfullDef[2];  // function provided by the port
                     String CapTyp=capfullDef[3];  //   Type   of   the   port   providing
capability
                     Class FctBlock=FunctionBlocks.get(Capfct);
                     //Step   3.2a:   Verifies   that   the   capacity   port   expects   a
specification flow
                     boolean fSpeCapExist = FctFlowExistance(Capfct,FctBlock,4);
                     if (fSpeCapExist){
                            // This paragraph is only for displaying the relationship
in a message box
                            printMe = CapSID + "["+CapPID+"] =: ";// FOR TESTING
                            printMe=printMe+"SpeFrom["+Capfct+"]"; // FOR TESTING
                            Message = Message + printMe+"\n";// FOR TESTING

                            // Step 3.2b: Generating MC variable names
                            ArrayList                                          <String[]>
Formulas=MCspelinkMaker(CapSID,CapPID,CapTyp, Capfct,StateRk,
                                          FunctionBlocks,varPathForSubsystem);

                            // Step 3.2c: Exporting formulas to MC
                            int varSize=Formulas.size();
                            for (int varN=0; varN<varSize;varN++){
                                   String[] Formulaterms=Formulas.get(varN);
                                   if
(specDefined.contains(Formulaterms[0])){Message=Message+"This     spec     was     defined
previously./n";
}
                                   else{//This spec is not yet defined
                                          specDefined.add(Formulaterms[0]);
                                          //MODEL CENTER
                                          mc.createLink(Formulaterms[0],
Formulaterms[1]);
                                          Message=Message+CapSID+":
"+Formulaterms[0]+"="+ Formulaterms[1]+"\n";// FOR TESTING
                                   }
                            }
                            Message=Message+"\n";
                     }// End of requirement spec flow to source
              }//End of scanning the capability ports
              if (debug){JOptionPane.showMessageDialog(null, Message);
} // FOR TESTING

              // Step 3.3: Realization of specification flows to requirement ports
              int  nReq  =  TableReq.size();  Message="Specification  flows  to  requirement
Ports:\n";
              for  (int  tarket  =  0;  tarket<nReq;  tarket++){  //This  loop  scan  all  the
capacity ports in architecture conf.
                     String[] reqfullDef =TableReq.get(tarket); // Load the identity of
the requirement
                     String reqSID=reqfullDef[0];  //  ID  of  the  system  imposing  the
requirement
                     String reqPID=reqfullDef[1];  //  ID  of  the  port  imposing  the
requirement
                     String reqfct=reqfullDef[2];  // function received by the port
                     String reqTyp=reqfullDef[3];  // Type of the requirement port
                     Class FctBlock=FunctionBlocks.get(reqfct);
                     //Step 3.2a: Verifies if the capacity port expects a specification
flow
                     boolean fSpeCapExist = FctFlowExistance(reqfct,FctBlock,3);
                     if ((fSpeCapExist)&&(reqTyp.equals("BoundaryFct")==false)){
                            // This paragraph is only for displaying the relationship
in a message box
                            printMe = reqSID + "["+reqPID+"] =: ";// FOR TESTING
                            printMe=printMe+"SpeFrom["+reqfct+"]"; // FOR TESTING
                            Message = Message + printMe+"\n";// FOR TESTING

                            // Step 3.2b: Generating MC variable names
                            ArrayList                                          <String[]>
```

```
Formulas=MCspelinkMaker(reqSID,reqPID,reqTyp, reqfct,StateRk,
                                        FunctionBlocks,varPathForSubsystem);

                                // Step 3.2c: Exporting formulas to MC
                                int varSize=Formulas.size();
                                for (int varN=0; varN<varSize;varN++){
                                        String[] Formulaterms=Formulas.get(varN);
                                        if
(specDefined.contains(Formulaterms[0])){Message=Message+"This     spec     was     defined
previously./n";
}
                                        else{//This spec is not yet defined
                                                specDefined.add(Formulaterms[0]);
                                                //MODEL CENTER
                                                mc.createLink(Formulaterms[0],
Formulaterms[1]);
                                                Message=Message+reqSID+":
"+Formulaterms[0]+"="+ Formulaterms[1]+"\n";// FOR TESTING
                                        }
                                }
                                Message=Message+"\n";
                        }// End of requirement spec flow to source
                }//End of scanning the capability ports
                if (debug){JOptionPane.showMessageDialog(null, Message);
 } // FOR TESTING

                // Saving modified MC model
                //MODEL CENTER
                mc.saveModelAs(filePathOut);

                return true;
        }// End of scanning elements in the block



        public  static  ArrayList  <String[]>  MCspelinkMaker(String  sID,String  pID,String
sType,String fctType,Integer stRk,
                        HashMap   <String,Class>   FunctionBlocks,   HashMap<String,String>
varPathForSubsystem){
                // Input definition:
                // 1: sID: Part name  with port                        2: pID: port name
                // 3: sType: Type of the part                4: fctType: Function assigned
on the port
                // 5: stRk : State Rank                                        6:
FunctionBlocks: Array containing references to the blocks describing the fcts
                // 7:varPathForSubsystem: Map which returns the variable path given the
type of the part


                ArrayList <String[]> Formulas = new ArrayList <String[]>();int varType =
0;
                if (pID.startsWith("Req")){varType=3;
}
                if (pID.startsWith("Cap")){varType=4;
}
                Class FctBlock=FunctionBlocks.get(fctType);
                // Importing name conventions
                ArrayList  <String[]>VarNames  =  MCVarNameConventions(fctType,  FctBlock,
varType);//Importing the the naming convention
                int varSize = VarNames.size();//the size of the variable implied by the
function
                String LinkTarget="";
                String Formula="";
                String PortIndex = pID.substring(3);
                for (int varRank=0;varRank<varSize;varRank++){ /* If the function implies
multiple variables
                        are passed this will build the formulas for all connections*/
                        //Target definition
                        String[] vName = VarNames.get(varRank);
                        if  (varType==4){//  This  specification  is  going  to  the  source
subsystem
                                String varPath="";
                                if
(varPathForSubsystem.containsKey(sType)){varPath=varPathForSubsystem.get(sType);
}
                                if (PortIndex.length()>0){ // The system provides more than
1 capability

        LinkTarget="Model."+sID+varPath+"."+vName[1]+PortIndex+"["+stRk+"]";//   Following
VarFormat 1
```

```
                                }else {
        LinkTarget="Model."+sID+varPath+"."+vName[1]+"["+stRk+"]";// Following VarFormat 1
                                }
                        }else if(varType==3){ // This specification is going to the load
side
                                if (sType.equals("BoundaryFct")){ //This condition checks
if the req is from a boundary function
                                        //  Functional  specs  are  ignored  for  boundary
functions
                                        varType=0;return null;
                                }else{
                                        String varPath="";
                                        if
(varPathForSubsystem.containsKey(sType)){varPath=varPathForSubsystem.get(sType);
}
                                        if (PortIndex.length()>0){ //  The system induces
more than 1 requirement
        LinkTarget="Model."+sID+varPath+"."+vName[2]+PortIndex+"["+stRk+"]";//   Following
VarFormat 2
                                        }else{
        LinkTarget="Model."+sID+varPath+"."+vName[2]+"["+stRk+"]";// Following VarFormat 2
                                        }
                                }
                        }// LinkTarget defined
                        if (varType!=0){
        Formula="Model.Mission."+fctType+"."+vName[3]+"["+stRk+"]";//TODO: Change this to
"_" rather than "."
                                String[] FullFormula={LinkTarget,Formula
};
                                Formulas.add(FullFormula);// Store in Map
                        }
                }
                return Formulas;
        }
        public static boolean FctFlowExistance(String fctType,Class fctBlock, int Type){
                /* Note: Type is a signal which can either be 1, 2 or 3.
                if Type == 1: We are looking for the variable convention for requirement
flows
                if  Type  ==  2:  We  are  looking  for  the  variable  convention  for
characteristic flows
                if Type == 3: We are looking for the variable convention for specification
flows to load
                if Type == 4: We are looking for the variable convention for specification
flows to source*/

                boolean fctFlowExists=false;

                // Step 1: search the elements to find the connectors representing the
variable flows
                for (Iterator <NamedElement> iter = fctBlock.getOwnedMember().iterator();
iter.hasNext(); ){
                        NamedElement namedElem = iter.next();
                        if (namedElem instanceof Connector){
                                Connector  conn  =  (Connector)  namedElem;  //  Stores
connector object in conn
                                // Step 2: Once a connector is retrieved , store the name
of the ports to which it is connected.
                                Iterator         <ConnectorEnd>         connEnds         =
conn.getEnd().iterator(); // The ends of the connector is stored in connEnds
                                ConnectorEnd connEnd1 = connEnds.next();
                                ConnectorEnd connEnd2 = connEnds.next();
                                String  Port1Type  =  connEnd1.getRole().getName();//Name  of
port on side 1;
                                String  Port2Type  =  connEnd2.getRole().getName();//Name  of
port on side 2;
                                //Step 3: Retrieve the direction of the information flow
                                String modelElement1="";String modelElement2="";
                                if
((Port2Type.equals("output"))&&(Port1Type.equals("input"))){//  NB  the  information  flow
goes
                                        //from an (model) output to a (model)input
                                        // Side 2 provides information to side 1
                                        modelElement1                                   =
connEnd2.getPartWithPort().getName();
                                        modelElement2                                   =
connEnd1.getPartWithPort().getName();
```

```
                                        //Step 4: Check the origin of the information flow
                                        if
((modelElement1.equals("load"))&&((modelElement2.equals("source")))&&(Type==1)){
                                                fctFlowExists = true;
                                        }else                                                   if
((modelElement1.equals("source"))&&((modelElement2.equals("load")))&&(Type==2)){
                                                fctFlowExists = true;
                                        }else                                                   if
((modelElement1.equals("architectureConcept"))&&((modelElement2.equals("load")))&&(Type==
3)){
                                                fctFlowExists = true;
                                        }else                                                   if
((modelElement1.equals("architectureConcept"))&&((modelElement2.equals("source")))&&(Type
==4)){
                                                fctFlowExists = true;
                                        }
                                }else
if((Port1Type.equals("output"))&&(Port2Type.equals("input"))){
                                       // Side 1 provides information to side 2
                                        modelElement1                                           =
connEnd1.getPartWithPort().getName();
                                        modelElement2                                           =
connEnd2.getPartWithPort().getName();
                                        //Step 5: Check the origin of the information flow
                                        if
((modelElement1.equals("load"))&&((modelElement2.equals("source")))&&(Type==1)){
                                                fctFlowExists = true;
                                        }else                                                   if
((modelElement1.equals("source"))&&((modelElement2.equals("load")))&&(Type==2)){
                                                fctFlowExists = true;
                                        }else                                                   if
((modelElement1.equals("architectureConcept"))&&((modelElement2.equals("load")))&&(Type==
3)){
                                                fctFlowExists = true;
                                        }else                                                   if
((modelElement1.equals("architectureConcept"))&&((modelElement2.equals("source")))&&(Type
==4)){
                                                fctFlowExists = true;
                                        }
                                }else{
                                        String nameFlow = "";
                                        for   (Iterator   <InformationFlow>   infoFlow   =
conn.get_informationFlowOfRealizingConnector().iterator();infoFlow.hasNext();){
                                                InformationFlow flow = infoFlow.next();

                                                Class         variableBlock=         (Class)
flow.getConveyed().iterator().next();
                                                nameFlow=variableBlock.getName();//  Records
the name of the variable
                                        }
                                       JOptionPane.showMessageDialog(null,"Error1:        A
variable flow was improperly defined in "+ fctType+"\nIllegal "+Port2Type+"<<"+Port1Type
+"direction on connection"+ nameFlow);
                                }
                        }
                }
                return fctFlowExists;
        }
        //TODO:functions  DefineVarPath  and  DefineVarArray  have  been  deactivated.  To
reactivate this functionality follow the instruction
         // in the comment started with the key word "WARNING"
         public        static        HashMap<String,String>        DefineVarPath(Property
partProperty,HashMap<String,String> varPathForSubsystem){

                // This method store the variable name path in MC in a map indexed by
subsystem type name
                String  SubsystemName  =  partProperty.getType().getName();//Type  of  the
subsystem implementing the requirement
                if (varPathForSubsystem.containsKey(SubsystemName)){
}
                else{
                        Collection<Element>                blockElements                =
partProperty.getType().getOwnedElement();//  Grabs  all  elements  defining  the  subsystems
considered
                        for   (Iterator   <Element>   elIter   =   blockElements.iterator();
elIter.hasNext(); ){
                                Element   thisElement   =   elIter.next();//This   scans   the
elements in the part

                                if (thisElement instanceof Property){//If the element is a
```

```
property
                                        Property valueProp = (Property) thisElement;//Cast
the proper class name...
                                        String                                          valueType=
valueProp.getType().getName();//... and get its type.

                                        // WARNING to reactivate this function you have to
define type in the magic draw environment and specify
                                        // the name of this type below (replace the "Error"
condition)
                                        if (valueType.equals("Error")){//If the type is
marked as an MC model reference...
                                                String
varNamePath=valueProp.getDefault();//...get the variable path

        varPathForSubsystem.put(SubsystemName,varNamePath);
                }}}}
                return varPathForSubsystem;
         }
        public       static       ArrayList       <String[]>       DefineVarArray(Property
partProperty,HashMap<String,String>        varPathForSubsystem,ArrayList        <String[]>
varPathArray){
                // This method store the variable namepath in MC in a map indexed by
subsystem type name
                String SubsystemName = partProperty.getType().getName();//Type of the
subsystem implementing the requirement
                if (varPathForSubsystem.containsKey(SubsystemName)){
}
                else{
                        Collection<Element>                blockElements                =
partProperty.getType().getOwnedElement();// Grabs all elements defining the subsystems
considered
                    for (Iterator <Element> elIter = blockElements.iterator();
elIter.hasNext(); ){
                            Element thisElement = elIter.next();//This scans the
elements in the part

                            if (thisElement instanceof Property){//If the element is a
property
                                    Property valueProp = (Property) thisElement;//Cast
the proper class name...
                                    String                                          valueType=
valueProp.getType().getName();//... and get its type.
                                    // WARNING to reactivate this function you have to
define type in the magic draw environment and specify
                                    // the name of this type below (replace the "Error"
condition)
                                    if (valueType.equals("Error")){//If the type is
marked as an MC model reference...
                                            String
varNamePath=valueProp.getDefault();//...get the variable path
                                            String[] thisOne={SubsystemName,varNamePath
};
                                            varPathArray.add(thisOne);
                }}}}
                return varPathArray;
         }
        public   static   ArrayList   <String[]>   MCreqlinkMaker(String   CapSID,String
CapPID,String CapType,ArrayList <String[]> ReqRel,Integer StateRank,
                        HashMap   <String,Class>   FunctionBlocks,   HashMap<String,String>
varPathForSubsystem, boolean debug,boolean showAll){
               // Detecting the function type carried by the link
               String[] reqRel=ReqRel.get(0);
               String Ftype = reqRel[4];
               ArrayList <String[]> Formulas = new ArrayList <String[]>();
               Class FctBlock=FunctionBlocks.get(Ftype);
               // Importing name conventions
               ArrayList   <String[]>VarName   =   MCVarNameConventions(Ftype,   FctBlock,
1);//Importing the the naming convention
               int varSize = VarName.size();//the size of the variable implied by the
function
               String LinkTarget="";
               String Formula="";
               String CapPortIndex = CapPID.substring(3);
               for (int varRank=0;varRank<varSize;varRank++){ /* If the function implies
multiple variables
                       are passed this will build the formulas for all connections*/
                       //Target definition
                       String[] vName = VarName.get(varRank);
                       String varPath="";
```

```
                        if
(varPathForSubsystem.containsKey(CapType)){varPath=varPathForSubsystem.get(CapType);
}
                        if (CapPortIndex.length()>0){ // The system provides more than 1
capability

       LinkTarget="Model."+CapSID+varPath+"."+vName[1]+CapPortIndex+"["+StateRank+"]";//
Following VarFormat 1
                        }else{

       LinkTarget="Model."+CapSID+varPath+"."+vName[1]+"["+StateRank+"]";//     Following
VarFormat 1
                        }

                        //Link formula definition
                        Formula="";
                        for (int rel=0;rel<ReqRel.size();rel++){
                                reqRel=ReqRel.get(rel);
                                String ReqSID=reqRel[0];
                                String ReqPID=reqRel[1];
                                String nRel=reqRel[2];
                                String ReqType = reqRel[3];
                                /* The requirement relationship is defined by:
                                [0]: ReqSID                             [1]: ReqPID
                                [2]: nRel                               [3]: ReqType
                                [4]: Function
                                 */
                                String ReqPortIndex = ReqPID.substring(3);
                                //  Definition  of  the  variable  name  containing  the
requirement
                                String ReqName="";
                                if (ReqType.equals("BoundaryFct")){ //This condition checks
if the req is from a boundary function
                                        if  (ReqPortIndex.length()>0){  //  The  boundary
function contains more than 1 requirement

       ReqName="Model.Mission."+vName[3]+ReqPortIndex+ReqSID+"["+StateRank+"]";//
Following VarFormat 3
                                        }else{

       ReqName="Model.Mission."+vName[3]+ReqSID+"["+StateRank+"]";// Following VarFormat
3
                                        }
                                }else{
                                        varPath="";
                                        if
(varPathForSubsystem.containsKey(ReqType)){varPath=varPathForSubsystem.get(ReqType);
}
                                        if (ReqPortIndex.length()>0){ // The system induces
more than 1 requirement

       ReqName="Model."+ReqSID+varPath+"."+vName[2]+ReqPortIndex+"["+StateRank+"]";//
Following VarFormat 2
                                        }else{

       ReqName="Model."+ReqSID+varPath+"."+vName[2]+"["+StateRank+"]";//       Following
VarFormat 2
                                        }
                                }
                                // Integrate the requirement in the formula
                                if  (nRel.equals("1")){//  if  100%  of  the  requirement  is
assigned to this link
                                        Formula=Formula+ReqName;//simply assign the variable
name to the formula
                                }else{//The requirement is split nRel times
                                        Formula=Formula+"1/"+reqRel[2]+"*"+ReqName;
                                }
                                if  (ReqRel.size()>rel+1){//  If  another  term  is  coming  in
this formula
                                        Formula=Formula+ "+";
                                }
                        }// done integrating all relationships to this link
                        String[] FullFormula={LinkTarget,Formula
};
                        if (showAll){JOptionPane.showMessageDialog(null, "Line 517 - Last
minute check: Show the formula\n"+LinkTarget+" = "+Formula);}
                        Formulas.add(FullFormula);// Store in Map
                        //JOptionPane.showMessageDialog(null,          "We're          at:
"+Formulas.size()+" out of "+varSize+".\nNote varRank="+varRank);//TESTING
                }
```

```
		return Formulas;
	}

	public static ArrayList <String[]> MCVarNameConventions(String fctType,Class fctBlock, int Type){
		/* Note: Type is a signal which can either be 1, 2 or 3.
		if Type == 1: We are looking for the variable convention for requirement flows
		if Type == 2: We are looking for the variable convention for characteristic flows
		if Type == 3: We are looking for the variable convention for specification flows to source
		if Type == 4: We are looking for the variable convention for specification flows to load*/

		ArrayList <String[]> Name=new ArrayList <String[]>();
		/* OUTPUT: This list is used to return the string contains the name convention:
		Each list element in the list contains the names for a given information flow.
		Each string array contains 4 strings:
		[0] Nature of the information passed (name of the block defining the flow)
		[1],[2] and [3] correspond to the naming convention for formats 1, 2 and 3 */

		// Step 1: search the elements to find the connectors representing the variable flows
		for (Iterator <NamedElement> iter = fctBlock.getOwnedMember().iterator(); iter.hasNext(); ){
			NamedElement namedElem = iter.next();

			if (namedElem instanceof Connector){
				boolean fctFlowExists=false;
				Connector conn = (Connector) namedElem; // Stores connector object in conn
				// Step 2: Once a connector is retrieved , store the name of the ports to which it is connected.
				Iterator <ConnectorEnd> connEnds = conn.getEnd().iterator(); // The ends of the connector is stored in connEnds
				ConnectorEnd connEnd1 = connEnds.next();
				ConnectorEnd connEnd2 = connEnds.next();
				String Port1Type = connEnd1.getRole().getName();//Name of port on side 1;
				String Port2Type = connEnd2.getRole().getName();//Name of port on side 2;
				//Step 3: Retrieve the direction of the information flow
				String modelElement1="";String modelElement2="";
				if ((Port2Type.equals("output"))&&(Port1Type.equals("input"))){// NB the information flow goes
					//from an (model) output to a (model)input
					// Side 2 provides information to side 1
					modelElement1 = connEnd2.getPartWithPort().getName();
					modelElement2 = connEnd1.getPartWithPort().getName();

					//Step 4: Check the origin of the information flow
					if ((modelElement1.equals("load"))&&((modelElement2.equals("source")))&&(Type==1)){
						fctFlowExists = true;
					}else if ((modelElement1.equals("source"))&&((modelElement2.equals("load")))&&(Type==2)){
						fctFlowExists = true;
					}else if ((modelElement1.equals("architectureConcept"))&&((modelElement2.equals("load")))&&(Type==3)){
						fctFlowExists = true;
					}else if ((modelElement1.equals("architectureConcept"))&&((modelElement2.equals("source")))&&(Type==4)){
						fctFlowExists = true;
					}
				}else if((Port1Type.equals("output"))&&(Port2Type.equals("input"))){
					// Side 1 provides information to side 2
					modelElement1 = connEnd1.getPartWithPort().getName();
					modelElement2 = connEnd2.getPartWithPort().getName();
```

```
                                        //Step 5: Check the origin of the information flow
                                        if
((modelElement1.equals("load"))&&((modelElement2.equals("source")))&&(Type==1)){
                                                fctFlowExists = true;
                                        }else                                                   if
((modelElement1.equals("source"))&&((modelElement2.equals("load")))&&(Type==2)){
                                                fctFlowExists = true;
                                        }else                                                   if
((modelElement1.equals("architectureConcept"))&&((modelElement2.equals("load")))&&(Type==
3)){
                                                fctFlowExists = true;
                                        }else                                                   if
((modelElement1.equals("architectureConcept"))&&((modelElement2.equals("source")))&&(Type
==4)){
                                                fctFlowExists = true;
                                        }
                                }else{
                                        String nameFlow = "";
                                        for    (Iterator    <InformationFlow>    infoFlow    =
conn.get_informationFlowOfRealizingConnector().iterator();infoFlow.hasNext();){
                                                InformationFlow flow = infoFlow.next();

                                                Class          variableBlock=          (Class)
flow.getConveyed().iterator().next();
                                                nameFlow=variableBlock.getName();//   Records
the name of the variable
                                        }
                                        JOptionPane.showMessageDialog(null,"Error2:        A
variable flow was improperly defined in "+ fctType+"\nIllegal "+Port2Type+"<<"+Port1Type
+"direction on connection"+ nameFlow);
                                }
                                if (fctFlowExists){
                                        // Step 6: Retrieve information conveyed
                                        for    (Iterator    <InformationFlow>    infoFlow    =
conn.get_informationFlowOfRealizingConnector().iterator();infoFlow.hasNext();){
                                                InformationFlow flow = infoFlow.next();
                                                String nameC[] = new String[4];
                                                Class          variableBlock=          (Class)
flow.getConveyed().iterator().next();
                                                nameC[0]=variableBlock.getName();//   Records
the description of the variable (block name)
                                                for   (Iterator   <NamedElement>   elemVars   =
variableBlock.getOwnedMember().iterator();elemVars.hasNext();){
                                                        NamedElement         elemVar         =
elemVars.next();// This loop access the value
                                                        if
(elemVar.getName().equals("Format1")){
                                                                Property          varProperty=
(Property) elemVar;
                                                                nameC[1]                        =
varProperty.getDefault();
                                                        }else
if(elemVar.getName().equals("Format2")){
                                                                Property          varProperty=
(Property) elemVar;
                                                                nameC[2]                        =
varProperty.getDefault();
                                                        }else
if(elemVar.getName().equals("Format3")){
                                                                Property          varProperty=
(Property) elemVar;
                                                                nameC[3]                        =
varProperty.getDefault();
                                                        }
                                                }
                                                Name.add(nameC);//  Add  the  variable  name
convention
                                        }
                                }
                        }
                }
                return Name;// Returns the name conventions detected in the function block
        }
        public     static     Boolean     FunctionalHandshake(String     PortLoadID,String
PortSourceID,String PortLoadType,String PortSourceType,String FunctionType){
                /* This method is true if the port names and functionality flow agree with
the naming convention*/
                if (PortSourceID.startsWith("Cap")){
                        if (PortLoadID.startsWith("Req")){
```

```
                                if
(PortSourceType.equals(FunctionType)&&PortLoadType.equals(FunctionType)){
                                        return true;
                                }else{
                                        String  Message="Definition  error:  The  functional
relationship impossible (functional missmatch between port types and connection).";
                                        Message = Message+"\nPortSourceID = "+PortSourceID+"
["+PortSourceType+"]\n           PortLoadID          =          "+          PortLoadID+"
["+PortSourceType+"]\n";JOptionPane.showMessageDialog(null, Message);
                                        JOptionPane.showMessageDialog(null, Message);
                                        return false;
                                }
                        }else{
                                String    Message="Definition    error:    PortSourceID    =
"+PortSourceID+"\n PortLoadID = "+ PortLoadID+"\n";
                                Message = Message+ " The functionality is not flowing from
a capability port.";
                                JOptionPane.showMessageDialog(null, Message);
                                return false;
                        }
                }else{
                        String Message="Definition error: PortSourceID = "+PortSourceID+"\n
PortLoadID = "+ PortLoadID+"\n";
                        Message = Message+ PortSourceID+" The functionality is not flowing
to a requirement port.";
                        JOptionPane.showMessageDialog(null, Message);
                        return false;
                }
        }
}
/* Variables naming conventions
------------------------------
*1- Contributing analysis names:
 * ----------------------------
 * CA representation of subsystems:
 * the name of the contributing analysis (CA) including the sizing model for a subsystem
 * must be the same as the name of the part property in the SysML model (no upper case)
 *
 * CA representation of boundary functions:
 * the boundary functions are defined in a CA named "Mission"
 *
 *2- Variables names:
 *-------------------
 *In MC the variables are designated by their full path. The path takes the following
form:
 *  Model.<name of the CA>.<name of the variable>[<Staterank>]
 *      <name of the CA> is defined above
 *      <name of the variable> is defined by:
 *              - The function characterized by the variable
 *              - By the port it is associated to based on:
 *                      - the port role (does it represent an induced req - i.e. output- or
 *                        a constraint on its capability - i.e. input)
 *                      - the rank of the port (in case the subsystem can have multiple
internal
 *                        component requesting or providing the functionality)
 *      <Staterank> is defined by:
 *              - TBD
 *
 *3- Number of elements in the functional link:
 *Depending on the function the variable name can be defined by different number of
double
 *(e.g. the electric power function could be defined by the Max power AND the nominal
transfert
 *or the fuel flow only requires the nominal flow transfert)
 *
 *
 * 3 Format for variables:
 * -----------------------
 * Format 1 (for capability variables - i.e. the side of the subsystem providing the
function)
 * Model.<Subsystem name>.<Name of variable implied by the function><Rank of the port if
any>[state rank]
 * Notes:
 *  - The name of the subsystem is contained in the part name of the SysML diagram
 *     -  The  name  of  variables  implied  by  the  function  is  stored  in  HashMap
<String,String[][]> MCVarNameConventions().
 * The name corresponding to format 1  is stored in [0][:].
 * - If several variables flows are implied by the function you have to go down that list
of variable names stored
 * in the 2nd dimension of MCVarNameConventions.
```

```
 *
 * Format 2 (for induced requirement variables - i.e. the side of the subsystem inducing
the requirement)
 * Model.<Subsystem part name>.<Name of variable implied by the function><Rank of the
port if any>[state rank]
 *  Note:  The  name  of  variables  implied  by  the  function  is  stored  in  HashMap
<String,String[][]> MCVarNameConventions().
 * The name corresponding to format 1  is stored in [1][:].
 *
 * Format 3 (for boundary requirement variables - i.e. the side of the subsystem inducing
the requirement)
 *  Model.Mission.<Name  of  variable  implied  by  the  function><Name  of  the  boundary
function>[state rank]
 * Note:
 *  -  The  name  of  variables  implied  by  the  function  is  stored  in  HashMap
<String,String[][]> MCVarNameConventions().
 * The name corresponding to format 1  is stored in [1][:].
 * - The name of the boundary function is contained in the part name of the SysML diagram

 */
```

I.4 ConfStructureMaker

```
// Modified on 0309 by Cyril
package edu.gatech.models.integration.mc;

import java.awt.event.ActionEvent;

import javax.swing.JOptionPane;

import com.nomagic.magicdraw.openapi.uml.SessionManager;
import com.nomagic.magicdraw.ui.browser.Node;
import com.nomagic.magicdraw.ui.browser.Tree;
import com.nomagic.magicdraw.ui.browser.actions.DefaultBrowserAction;
import com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class;
import com.phoenix_int.ModelCenter.ModelCenter;
import com.phoenix_int.ModelCenter.ModelCenterException;

public class ConfStructureMaker extends DefaultBrowserAction
{
	private static final long serialVersionUID = 1L;

	public ConfStructureMaker()
	{
		super("", "Test configuration", null, null);
	}

	public void actionPerformed(ActionEvent e)
	{
		Tree tree = getTree();
		for (Integer i = 0; i < tree.getSelectedNodes().length; i++)	// This
loop scans all nodes selected in the MD browser tree
		{
			Node node = tree.getSelectedNodes()[i];
			Object userObject = node.getUserObject();
			if (!SessionManager.getInstance().isSessionCreated())
			{
				SessionManager.getInstance().createSession("Configuration
Action");
			}
			if (userObject instanceof Class){    //Action  will  be  applied  to
elements of type "Class" (which includes SysML blocks)
				Object[]  optionsShowDebug  =  {"View  conclusions","Show  all
steps"
};
				int  nShowDebug=JOptionPane.showOptionDialog(null,  "Choose
from option",
					"Proceed?",	JOptionPane.YES_NO_OPTION,
JOptionPane.QUESTION_MESSAGE, null, optionsShowDebug, optionsShowDebug[0]);
				boolean debugMode=true;
				boolean showAll=false;
				if(nShowDebug==1){
					showAll=true;}

				String filePathIn = JOptionPane.showInputDialog(null, "File
Path  Name",  "C:\\OfficeFolder\\ModelGeneration\\TestComposition.pxc");//TODO:  Move  this
around if needed...
```

```
                                String
filePathOut="C:\\OfficeFolder\\ModelGeneration\\TestStructure.pxc";//FOR TESTING
                                Integer StateRk = 0;//FOR TESTING
                                Class ConfBlock = (Class) userObject;
                                ModelCenter mc;
                                try {
                                        mc = new ModelCenter();
                                        ConfStructureHelper.ConfStructureBuilder(ConfBlock,
StateRk, filePathIn, filePathOut,debugMode, showAll, mc);
                                } catch (ModelCenterException e1) {
                                        JOptionPane.showMessageDialog(null,
e1.getMessage());
                                }

                        }else{JOptionPane.showInputDialog(null,"This doesn't seem to be a
configuration");
}
                }
                SessionManager.getInstance().closeSession();
        }
        public void updateState(){
                if (getTree().getSelectedNodes().length > 1)
                        this.setEnabled(false);
        }
}
```

I.5 ScenarioDefinition

```
package edu.gatech.models.integration.mc;

import java.awt.event.ActionEvent;
import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;
import java.util.Iterator;

import javax.swing.JOptionPane;

import com.nomagic.magicdraw.openapi.uml.SessionManager;
import com.nomagic.magicdraw.ui.browser.Node;
import com.nomagic.magicdraw.ui.browser.Tree;
import com.nomagic.magicdraw.ui.browser.actions.DefaultBrowserAction;
import com.nomagic.uml2.ext.magicdraw.activities.mdbasicactivities.ActivityEdge;
import com.nomagic.uml2.ext.magicdraw.activities.mdbasicactivities.ActivityFinalNode;
import com.nomagic.uml2.ext.magicdraw.activities.mdbasicactivities.InitialNode;
import com.nomagic.uml2.ext.magicdraw.activities.mdfundamentalactivities.Activity;
import com.nomagic.uml2.ext.magicdraw.activities.mdfundamentalactivities.ActivityNode;
import com.nomagic.uml2.ext.magicdraw.classes.mddependencies.Dependency;
import com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class;
import com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element;
import com.nomagic.uml2.ext.magicdraw.classes.mdkernel.InstanceValue;
import com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Slot;
import com.nomagic.uml2.ext.magicdraw.classes.mdkernel.ValueSpecification;
import com.phoenix_int.ModelCenter.ModelCenter;
import com.phoenix_int.ModelCenter.ModelCenterException;

public class ScenarioDefinition extends DefaultBrowserAction{

        private static final long serialVersionUID = 1L;

        public ScenarioDefinition()
        {
                super("", "Scenario testing and architecture building", null, null);
        }

        public void actionPerformed(ActionEvent e)
        {
                if (!SessionManager.getInstance().isSessionCreated())
                {
                        SessionManager.getInstance().createSession("Integration Action");
                }
                Tree tree = getTree();
                for (Integer i = 0; i < tree.getSelectedNodes().length; i++){      // This
loop scans all nodes selected in the MD browser tree
                        Node node = tree.getSelectedNodes()[i];
```

```
                        Object userObject = node.getUserObject();
                        if (userObject instanceof Activity){ //Action will be applied to
elements of type "Activity"


                                ////////////////////////////////////////////////////
                                //             Importing scenario def from SysML
        //
                                ////////////////////////////////////////////////////


                                //Definition of main variables
                                HashMap   <String,String[]>   scenariosMap=   new   HashMap
<String,String[]>();
                                // The element in the scenariosMap are tagged by scenario
name. The information contained in the String[] define:
                                // [0]- The configuration used for the scenario
                                //  [1]-  The  criticality  level  used  to  defined  the
requirements
                                // [2]- The mission phase

                                ArrayList<String[]>     transitions=     new     ArrayList
<String[]>();
                                // This array lists the transitions between scenarios in
the graph. String[] containts:
                                //[0]- The scenario up (before the transition)
                                //[1]- The scenario down (after the transition)

                                // Variables used for tree processing
                                HashMap   <String,Integer>   scenariosRank   =   new   HashMap
<String,Integer>();//Rank in transition matrix
                                HashMap   <Integer,String>   rankScenarios   =   new   HashMap
<Integer,String>();//Rank in transition matrix
                                Integer rankToStart = null;// Variable pointing toward the
starting point in the transition matrix
                                //Integer rankToEnd = null;// Variable pointing toward the
ending point in the transition matrix

                                Activity element = (Activity) userObject;

                                // Keeping track of the configuration blocks
                                HashMap   <String,Class>   configClassMap=   new   HashMap
<String,Class>();//Ref to class
                                // Keeping track of the start and final nodes
                                ArrayList  <String>  startNodes=  new  ArrayList  <String>
();//records the names of initial nodes
                                ArrayList   <String>   endNodes=   new   ArrayList   <String>
();//records the names of final nodes
                                String  endMarker="end";//Universal  name  refering  to  the
ending node


                                // Choosing display options
                                ////////////////////////////

                                Object[] optionsShowDebug = {"Test mission sequences","Test
sequences and display each step","Build the architecture now"
};
                                int  nShowDebug=JOptionPane.showOptionDialog(null,  "Choose
from option",
                                                "Proceed?",        JOptionPane.YES_NO_OPTION,
JOptionPane.QUESTION_MESSAGE, null, optionsShowDebug, optionsShowDebug[0]);
                                JOptionPane.showMessageDialog(null,nShowDebug);
                                boolean ScenarioTestingOnly=true;
                                boolean displayScenarioInfo=false;
                                boolean Debug=true;
                                if (nShowDebug==0){ScenarioTestingOnly=true;Debug=true;
}
                                if
(nShowDebug==1){ScenarioTestingOnly=true;Debug=true;displayScenarioInfo=true;
}
                                else
{ScenarioTestingOnly=false;Debug=false;displayScenarioInfo=false;
}

                                boolean errorScenarioReading = false; // Internal error was
identified. The process will stop.

                                Integer nodeTypeRank=0;
                                for        (Iterator        <Element>        iter        =
```

```
element.getOwnedElement().iterator(); iter.hasNext(); ){
                                    Element namedElem = iter.next();

                                    if ((namedElem instanceof InitialNode)){
                                                        // STARTING NODE
                                                        InitialNode    iNode=    (InitialNode)
namedElem; // recasting correct variable class
                                                        String iNodeName= iNode.getName();

        if(iNodeName.length()==0){iNodeName="start";iNode.setName("start");
}
                                                        startNodes.add(iNodeName);
                                                        // Detecting the edges coming out of
the initial node
                                                        Collection <ActivityEdge> linksOut =
iNode.getOutgoing();int nConn=linksOut.size();
                                                        for      (Iterator      <ActivityEdge>
linkn=linksOut.iterator(); linkn.hasNext(); ){
                                                                //Grab the links out
                                                                ActivityEdge
link=linkn.next();
                                                                ActivityNode
nodeDown=link.getTarget();
                                                                String      nameNodeDown      =
nodeDown.getName();

        if(nameNodeDown.length()>0){// If this link is not terminated don't include it.
                                                                        String[]
linkdefinition={iNodeName,nameNodeDown
};

        transitions.add(linkdefinition);
                                                        }}

        scenariosRank.put(iNodeName,nodeTypeRank);rankScenarios.put(nodeTypeRank,
iNodeName);
                                                        rankToStart=nodeTypeRank;
                                                        if (displayScenarioInfo) {

        JOptionPane.showMessageDialog(null, "Scenario Name: "+iNodeName+
                                                                "\nScenarioRank:
"+nodeTypeRank+"\nConfiguration: "+"N.A."+"\n"+
                                                                nConn);//FOR TESTING
                                                        }
                                                        nodeTypeRank++;
                                                        String[] scenarioDef={"","","0"
};

        scenariosMap.put(iNodeName,scenarioDef);


                                    }else if ((namedElem instanceof ActivityFinalNode)){
                                                        //FINAL NODE
                                                        ActivityFinalNode               fNode=
(ActivityFinalNode) namedElem; // recasting correct variable class
                                                        String fNodeName= fNode.getName();

        if(!fNodeName.equals(endMarker)){fNodeName=endMarker;fNode.setName(endMarker);
}
                                                        endNodes.add(fNodeName);

        scenariosRank.put(fNodeName,nodeTypeRank);
                                                        rankScenarios.put(nodeTypeRank,
fNodeName);
                                                        if (displayScenarioInfo) {

        JOptionPane.showMessageDialog(null, "Scenario Name: "+fNodeName+
                                                                "\nScenarioRank:
"+nodeTypeRank+"\nConfiguration: "+"N.A."+"\n"+
        scenariosRank.size()+"/"+rankScenarios.size());//FOR TESTING
                                                        }
                                                        nodeTypeRank++;
                                                        String[] scenarioDef={"","","0"
};

        scenariosMap.put(fNodeName,scenarioDef);


                                    }else if ((namedElem instanceof ActivityNode)){
```

```
                                                    // Accessing an activity node
                                                    ActivityNode scenario= (ActivityNode)
namedElem;// recasting correct variable class
                                                    String         scenarioName         =
scenario.getName();
                                                    String[] scenarioDef={"","",""
};String Message="";
                                                    String nameConfig="";
                                                    // Verify that the scenario was not
already defined
                                                    if
(scenariosRank.containsKey(scenarioName)){
       JOptionPane.showMessageDialog(null, "Warning Scenario definition: \nThe scenario
named ["+scenarioName+
                                                                                "] was defined
more than once. Its second definition will not be considered for the model."+"\n"+
                                                                                "To     prevent
this warning to arise in the future delete the redundant definition in the tree.");
                                                    }else{
                                                           //   Detection   of   outbound
connections
                                                           Collection      <ActivityEdge>
linksOut = scenario.getOutgoing();int nConn=linksOut.size();
                                                           for  (Iterator  <ActivityEdge>
linkn=linksOut.iterator(); linkn.hasNext(); ){
                                                                  ActivityEdge
link=linkn.next();
                                                                  ActivityNode
nodeDown=link.getTarget();
                                                                  String  nameNodeDown  =
nodeDown.getName();
       if(nameNodeDown.length()>0){// If this link is not terminated don't include it.
                                                                                String[]
linkdefinition={scenarioName,nameNodeDown
};
       transitions.add(linkdefinition);
                                                           }}
                                                           //    Detection    of    the
configuration used in this scenario
                                                           Collection        <Dependency>
configurations=scenario.getClientDependency();
                                                           for   (Iterator   <Dependency>
confn=configurations.iterator(); confn.hasNext(); ){
                                                                  Dependency
conf=confn.next();
                                                                  Element
targetconf=conf.getTarget().iterator().next();//The  relevant  configuration  will  be  in
this nodeUp object
                                                                  Class config= (Class)
targetconf;
                                                                  nameConfig           =
config.getName();
       configClassMap.put(nameConfig, config);
       scenarioDef[0]=nameConfig;
                                                                  }
                                                           //  Detection  of  the  flight
phase and criticality
                                                           Collection  <Slot>  slots  =
scenario.getAppliedStereotypeInstance().getSlot();
                                                           for (Iterator <Slot> slotiter
= slots.iterator(); slotiter.hasNext();){
                                                                  //grabs    the    slots
(i.e. tags) of the element
                                                                  Slot
slot=slotiter.next();
                                                                  String
characteristicName = slot.getDefiningFeature().getName();// Slot category name
                                                                  ValueSpecification
characteristicValue= slot.getValue().iterator().next();
                                                                  if
(characteristicValue instanceof InstanceValue){
```

```
                                                                        String value =
((InstanceValue) characteristicValue).getInstance().getName();//Value stored in the slot
                                                                        if
(characteristicName.equals("criticality")){

        scenarioDef[1]=value;
                                                                        }else
if(characteristicName.equals("flightPhase")){

        scenarioDef[2]=value;
                                                                        }

        Message=Message+characteristicName+": "+value+"\n";//FOR TESTING
                                                                    }}


        scenariosRank.put(scenarioName,nodeTypeRank);rankScenarios.put(nodeTypeRank,
scenarioName);
                                                                    if (displayScenarioInfo) {

        JOptionPane.showMessageDialog(null, "Scenario Name: "+scenarioName+
                                                                            "\nScenarioRank:
"+nodeTypeRank+"\nConfiguration: "+nameConfig+"\n"+
                                                                            nConn);//FOR TESTING
                                                                            }
                                                                    nodeTypeRank++;

        scenariosMap.put(scenarioName,scenarioDef);
                                                            }

                                    }// Done with this activity node
                                } //Done importing from SysML

                                //+++  DISPLAY AND VERIFICATION SECTION  +++//

                                // List the detected connections and check the validity of
its destination:
                                ArrayList<String[]>  transitionsValidated=  new  ArrayList
<String[]>();
                                String Message="\nThe links detected are:";
                                for (int n=0;n<transitions.size();n++){
                                    //for             (Iterator              <String[]>
linkn=transitions.iterator();linkn.hasNext();){
                                    String[]link=transitions.get(n);//linkn.next();
                                    if  (scenariosRank.containsKey(link[1])){//  If  the
destination was properly defined
                                        Message=Message+"\n"+link[0]+">>"+link[1];
//This stores the link in the connection to display at the end

        transitionsValidated.add(transitions.get(n));
                                    }else{
                                        String  Display=  "Warning:  The  transition
from "+link[0]+" to "+link[1]+
                                        "can  not  be  created  because  the  scenario
"+link[1]+" does not exist.";
                                        Object[]  optionsGhostTransition  =  {"Ignore
connection","Stop the routine"
};
                                        int
decisionOnGhost=JOptionPane.showOptionDialog(null,Display,
                                                        "Proceed?",
JOptionPane.YES_NO_OPTION,  JOptionPane.QUESTION_MESSAGE,  null,  optionsGhostTransition,
optionsGhostTransition[0]);

        if(decisionOnGhost==1){errorScenarioReading=true;
}
                                }}// Done checking the validity of transitions

                                if
((Debug)&&(errorScenarioReading==false)){JOptionPane.showMessageDialog(null,"The  size  of
the connector array is: "+transitionsValidated.size()+Message);
}

                                // List the detected nodes with their rank
                                Message ="\n";
                                int nNodes = scenariosRank.size();
                                int nRanks = scenariosRank.size();
                                String overview = "There are "+nNodes+" stored in "+nRanks;
                                for (int nodeR=0;nodeR<nNodes;nodeR++){
                                    String ScenName=rankScenarios.get(nodeR);
```

```
                                    Message = Message+"\n"+ScenName+" at rank "+nodeR+"
(verif rank "+scenariosRank.get(ScenName)+")";
                                }
                                if
((Debug)&&(errorScenarioReading==false)){JOptionPane.showMessageDialog(null,overview+Mess
age);
}


                                // Verification of the existance of a start and end node
                                String Message1 = "Error: The graph does not include a
";String Message2 = " node. \n Make sure you have no duplicate.";
                                String Message3 = "Error: The graph includes more than 1 ";
                                int nInitial = startNodes.size();
                                errorScenarioReading=(nInitial==0);// There are no starting
points
                                if
(errorScenarioReading){JOptionPane.showMessageDialog(null,Message1+"starting"+Message2);
}
                                // There are more than 1 starting nodes
                                else
if(nInitial>1){errorScenarioReading=true;JOptionPane.showMessageDialog(null,Message3+"end
nodes");
}

                                int nEnd = endNodes.size();
                                errorScenarioReading=(nEnd==0);//   There   are   no   ending
points
                                if
(errorScenarioReading){errorScenarioReading=true;JOptionPane.showMessageDialog(null,Messa
ge1+"end"+Message2);
}
                                else
if(nEnd>1){errorScenarioReading=true;JOptionPane.showMessageDialog(null,Message3+"end
nodes");
}

                                if (!errorScenarioReading){









        ////////////////////////////////////////////////////////////
                                //                                    Defining     the     possible
sequences                       //

        ////////////////////////////////////////////////////////////

                                //Building linkMatrix
                                //  the  linkMatrix  represents  the  interconnections  in  a
matrix form
                                // if linkMatrix[i][j]=1
                                int[][]linkMatrix                                          =
ConnectionMatrixBuilder(transitionsValidated,scenariosRank);

                                // Building sequences
                                //////////////////////

                                // If in testing mode ask if the steps should be displayed
                                boolean giveChoice=false;
                                if (nShowDebug==1){giveChoice=true;
}
                                ArrayList                <String[]>                seqArray
=SequenceDetector(linkMatrix,scenariosRank,rankScenarios,rankToStart,endMarker,giveChoice
);
                                if(Debug){DisplaySeqArraySimple(seqArray);
}


                                // Definition of overall sequence of states
                                ////////////////////////////////////////////

                                ArrayList                                                 <String[]>
scenariosSchedule=ScenariosScheduler(seqArray, scenariosMap);
                                // The element in the scenariosSchedule list a series of
sequences. The information contained in the String[] define:
```

```
                                // [0]- Name of the scenario
                                // [1]- The configuration used for the scenario
                                //  [2]-  The  criticality  level  used  to  defined  the
requirements
                                // [3]- The mission phase
                                // [4]- Reinitiation variable (0 if normal scenario, 1 if
placeholder for restart)
                                int totalNstates = scenariosSchedule.size();









                                if(!ScenarioTestingOnly){//This  condition  deactivates  the
building of the architecture

        ////////////////////////////////////////////////////////////////
                                //
        Building Model                                          //

        ////////////////////////////////////////////////////////////////

                                String        filePathIn="C:\\Program        Files\\Phoenix
Integration\\Analysis          Server          5.1\\analyses\\Archsetup\\MC          Start
files\\"+JOptionPane.showInputDialog(null,"Enter        file        name        (without
extention):","Start1")+".pxc";
                                String
filePathOut="C:\\OfficeFolder\\ModelGeneration\\Composition.pxc";

                                // Importing composition
                                //--------------------------
                                boolean CompOK = true;
                                //Step 1: Read Architecture block from SysML
                                Collection                                          <Dependency>
dependencyOnActivity=element.getClientDependency();  //  Referring  to  the  architecture
block for importing the composition

                                for                  (Iterator                      <Dependency>
dependN=dependencyOnActivity.iterator(); dependN.hasNext(); ){
                                        //Grab the configuration out
                                        Dependency dependentComposition=dependN.next();
                                        Element
architectureElement=dependentComposition.getTarget().iterator().next();//The     relevant
configuration will be in this nodeUp object
                                        Class           architectureBlock=           (Class)
architectureElement;// This is the class which describes the architecture composition

                                        //Step  2:  Import  and  create  instances  of  relevant
model in MC

                                        if
(Debug){JOptionPane.showMessageDialog(null,"Launch Composition Maker");
}
                                        CompOK
=CompositionMaker.CompositionBuilder(architectureBlock,     totalNstates,     filePathIn,
filePathOut, false, Debug);// NB this method initializes the DV
                                        filePathIn=filePathOut;
                                }

                                if(CompOK){


                                // Importing structure
                                //--------------------


                                //ArrayList   <String[]>   scenariosSchedule=new   ArrayList
<String[]> ();
                                // The element in the scenariosSchedule list a series of
sequences. The information contained in the String[] define:
                                // [0]- Name of the scenario
                                // [1]- The configuration used for the scenario
                                //  [2]-  The  criticality  level  used  to  defined  the
requirements
                                // [3]- The mission phase
                                // [4]- Reinitiation variable (0 if normal scenario, 1 if
placeholder for restart)
```

```
                                // Importing conf by conf
                                //-----------------------
                                Object[]     optionsInterface2     =     {"Proceed     with
structure","Bypass structure"
};

                                int    structureA=JOptionPane.showOptionDialog(null,    "The
builder is about to iterate on states and construct the structure",
                                                "Proceed?",        JOptionPane.YES_NO_OPTION,
JOptionPane.QUESTION_MESSAGE, null, optionsInterface2, optionsInterface2[0]);

                                if (structureA==0){
                                        try {
                                                ModelCenter mc = new ModelCenter();
                                                boolean                                  BFsetup=
BoundaryFunctionBuilder.ExportBF(scenariosSchedule, filePathIn, filePathOut, mc);
                                                //boolean
BFsetup=BoundaryFunctionBuilder.ExportBF(scenariosSchedule,   filePathOut,   filePathOut,
mc);
                                                if (BFsetup){
                                                        boolean ConfOK=true;
                                                        for                                  (int
state=0;(state<totalNstates)&&(ConfOK);state++){// Proceed to state by state
                                                                String[]
scDesc=scenariosSchedule.get(state);//Retrieve the scenario description
                                                                if (Debug){

        JOptionPane.showMessageDialog(null,"Launching StructureMaker on rank: "+state+
                                                                                "\nScenario:
"+scDesc[0]+

        "\nConfiguration: "+scDesc[1]+

        "\nCriticality: "+scDesc[2]+
                                                                                "\nPhase:
"+scDesc[3]+
                                                                                "\nNew
sequence: "+scDesc[4]);
                                                                }
                                                                if       (state==totalNstates-
1){filePathOut="C:\\OfficeFolder\\ModelGeneration\\FinalModel.pxc";
}

        else{filePathOut="C:\\OfficeFolder\\ModelGeneration\\Temp\\"+state+".pxc";
}
                                                                if  (scDesc[4].equals("1")){//
If this state corresponds to a reinitialization...
                                                                        //...do nothing
                                                                }else{//Otherwise...
                                                                        String
configName=scDesc[1];//... retrieve the configuration name...
                                                                        Class
configBlock=configClassMap.get(configName);//... retrieve the configuration description
                                                                        if (Debug){
                                                                                int
stateDisp=state+1;

        JOptionPane.showMessageDialog(null,"Launch        Structure        Maker        for
"+stateDisp+"/"+totalNstates);
                                                                        }

                                                                        //Build the functional
relationships for each scenario

        ConfOK=ConfStructureHelper.ConfStructureBuilder(configBlock,   state,   filePathIn,
filePathOut,false, false, mc);

        filePathIn=filePathOut;
                                                                }
                                                        }
                                                        if
(ConfOK){JOptionPane.showMessageDialog(null,"Done and releasing the mc model");
 }

        else{JOptionPane.showMessageDialog(null,"An  error  occured  while  importing  the
structure");
}
                                                }else{JOptionPane.showMessageDialog(null,"An
error occured while importing the boundary function parameters");
}
```

```
                                        mc.release();

                                    } catch (ModelCenterException e1) {
                                        JOptionPane.showMessageDialog(null,
e1.getMessage());
                                    }
                                }
                                }else{JOptionPane.showMessageDialog(null,"An  error  occured
while importing the composition");
}
                                }else{}//No composition or structure construction requested
by the user (i.e. ScenarioTestingOnly is true)
                                }else{JOptionPane.showMessageDialog(null,"An  error  occured
while importing the scenarios");
}
                        }

                }
                SessionManager.getInstance().closeSession();
        }

        public void updateState(){
                if (getTree().getSelectedNodes().length > 1)
                        this.setEnabled(false);
        }
        public    static    void    DisplayNextNode(ArrayList<String>    nextNodes,String
currentNode){
                String Message="Current node: "+currentNode+ "\nList of next nodes:\n";
                int  nNext=nextNodes.size();
                for (int n=0;n<nNext;n++){
                        Message=Message+nextNodes.get(n)+"\n";
                }
                JOptionPane.showMessageDialog(null,Message);
        }

        public static String DisplaySequence(String[] Seq){
                int lengthSeq=Seq.length;int last=0;
                if (Seq[lengthSeq-1].equals(null)){last=lengthSeq-1;
                }else{last=lengthSeq;
}
                String Message="";
                for(int n=0;n<last;n++){
                        Message = Message + Seq[n];
                        if (n<last){Message = Message + " >> ";
}
                }
                return Message;
        }
        public static void DisplaySeqArraySimple(ArrayList <String[]> seqArray){
                // Method displaying the sequences as they stand
                String Message="The possible sequences are:";
                for (int seq=0;seq<seqArray.size();seq++){
                        String[] thisSequence=seqArray.get(seq);
                        Message=Message+"\n Seq.#"+seq+": ";
                        for(int n=0;n<thisSequence.length;n++){
                                Message = Message + thisSequence[n];
                                if (n<thisSequence.length){Message = Message + " >> ";
}
                        }
                }
                JOptionPane.showMessageDialog(null,Message);
        }
        public static boolean DisplaySeqArray(ArrayList <String[]> seqArray, int iter, int
activNode, int activSeq, String prefix){
                // Method displaying the sequences as they stand
                String  Message=prefix+  "Active  sequence:  "+activSeq+"/"+seqArray.size()+
"\nThe active scenario: "+activNode;
                for (int seq=0;seq<seqArray.size();seq++){
                        String[] thisSequence=seqArray.get(seq);
                        Message=Message+"\n Seq.#"+seq+": ";
                        for(int n=0;n<thisSequence.length;n++){
                                Message = Message + thisSequence[n];
                                if (n<thisSequence.length){Message = Message + " >> ";
}
                        }
                }
                Object[] optionsShowDebug = {"Go to next step","Skip to end"
};
                int nShowSteps=JOptionPane.showOptionDialog(null, Message,
                                "Proceed?",                         JOptionPane.YES_NO_OPTION,
```

```
JOptionPane.QUESTION_MESSAGE, null, optionsShowDebug, optionsShowDebug[0]);

                return (nShowSteps==0);
        }


        public static ArrayList <String[]> SequenceDetector(int[][]linkMatrix, HashMap
<String,Integer> scenariosRank,
                        HashMap    <Integer,String>    rankScenarios,int    rktoStart,String
endMarker, boolean giveChoice){
                // Inputs:
                // linkMatrix: Matrix specifying connections from element on the row to
the element on column
                // scenariosRank: Map specifying rank based on scenario
                // rankScenarios: Map specifying scenario based on rank
                // rktoStart = rank of the initial nodes
                ArrayList <String[]> seqArray = new ArrayList <String[]> ();

                boolean checkProgress=false;
                if (giveChoice){
                        Object[] optionsShowSeq = {"No","Yes"
};
                        int  nShowSeq=JOptionPane.showOptionDialog(null,  "Do  you  want  to
monitor the sequences as they get built",
                                        "Proceed?",               JOptionPane.YES_NO_OPTION,
JOptionPane.QUESTION_MESSAGE, null, optionsShowSeq, optionsShowSeq[0]);
                        if(nShowSeq==1){checkProgress=true;
}
                }
                int activSeq = 0; // Current sequence rank (within the seqArray)
                int  activNode   = 0;// Current position in the current sequence (within
String[] in the seqArray)
                int Nnodes = scenariosRank.size();

                // Initialization
                String currentNode = "start"; // This variable contains the name of the
activNode in the activSeq.
                int  nCurrent  =  rktoStart;  //  This  variable  indicates  the  rank  inside
linkMatrix which lists the
                // connections to currentNode
                String[] sequenceStarter = {currentNode
 };// Initialization of the first sequence
                seqArray.add(sequenceStarter);// Integration of the first sequence in the
array of sequences
                boolean moreComing = true;// This describe the status on the convergence
condition for exploring the tree
                int maxIter=100;int iter=0;// just in case, there are a maximum number of
iteration for the exploration

                while((moreComing)&&(iter<maxIter)) {// Loop exploring the tree
                        // Step 1:
                        ArrayList<String> nextNodes = new ArrayList<String>();
                        for (int n=0;n<Nnodes;n++){//Scans nodes one by one
                                if (linkMatrix[nCurrent][n]==1){// if the node is connected
                                        nextNodes.add(rankScenarios.get(n));//  the  sequence
may proceed with that node
                                }
                        }// At this point we have listed the possible next nodes to the
current sequence
                        //DisplayNextNode(nextNodes,currentNode);//FOR TESTING


                        String Message="";
                        // Step 2:
                        String[]  initSeq  =  seqArray.get(activSeq);  //Registers  previous
nodes in the current sequence
                        // Create a new string of scenario with an additional space for the
additional node
                        String[]  upSeq  =  new  String[activNode+2];for  (int  n  =  0;
n<activNode+1; n++){upSeq[n]=initSeq[n];
}
                        Message="InitialString: ";
                        Message=Message+DisplaySequence(initSeq)+"\n";
                        //Step 3.1: Update the current sequence
                        upSeq[activNode+1]=nextNodes.get(0);// Add the next node...
                        Message=Message+"The next node will be: "+nextNodes.get(0)+"\n";
                        Message=Message+"The         possible         sequences         are:
\n"+DisplaySequence(upSeq)+"\n";
                        if(checkProgress){checkProgress=DisplaySeqArray(seqArray,     iter,
activNode, activSeq, "Before addition\n");
```

```
}
                        seqArray.add(activSeq,upSeq);seqArray.remove(activSeq+1);//...and
update the current sequence

                        if (nextNodes.size()>1){
                                //Step 3.2: Branching (i.e. there are several possible next
nodes)
                                for (int n=1;n<nextNodes.size();n++){
                                        String[] upSeq2 = new String[activNode+2];for (int
n1 = 0; n1<activNode+1; n1++){upSeq2[n1]=initSeq[n1];
}
                                        upSeq2[activNode+1]=nextNodes.get(n);//    Add    the
other possible node...
                                        seqArray.add(upSeq2);//...and   update   the   current
sequence
                                        Message=Message+DisplaySequence(upSeq2)+"\n";
                                }
                        }else{
                                //DisplaySeqArray(seqArray,  iter,  activNode,  activSeq,"No
branching necessary\n");//FOR TESTING
                        }
                        //JOptionPane.showMessageDialog(null,Message);

                        //Step 4: Shift the position of the pointer
                        currentNode=nextNodes.get(0);
                        nCurrent=scenariosRank.get(currentNode);

                        // Step 5: Check if we have reached the end of the current sequence
                        if (currentNode.equals(endMarker)){
                                while((currentNode.equals(endMarker))&&(moreComing)){
                                        //Step 6: Move on to the next sequence
                                        activSeq++;

                                        //Step 7: Checks that the next sequence exists
                                        if (activSeq<seqArray.size()){

                                                //Step 8: Move to the last node of the next
sequence
                                                String[] newSeq=seqArray.get(activSeq);
                                                activNode=newSeq.length-1;
                                                currentNode = newSeq[activNode];
                                                nCurrent=scenariosRank.get(currentNode);
                                        }else{
                                                //Step 9: All sequences were completed
                                                moreComing=false;

        //JOptionPane.showMessageDialog(null,"Exploration complete!!!");
                                                //DisplaySeqArray(seqArray, iter, activNode,
activSeq,"");//FOR TESTING
                                        }
                                }
                        }else{ // The sequence was incomplete: proceed to the next node
                                activNode++;
                                moreComing=true;
                        }
                        //DisplaySeqArray(seqArray,  iter,  activNode,  activSeq,"");//FOR
TESTING
                        iter++;
                }
                if (iter==maxIter){
                        String  Wmessage="Warning  sequence  builder:\nMore  sequences  may
exist";
                        Wmessage=Wmessage+"\nMake sure that no loops are present in the
graph.";
                        JOptionPane.showMessageDialog(null,Wmessage);
                }
                return seqArray;
        }
        public       static       int[][]       ConnectionMatrixBuilder(ArrayList<String[]>
transitions,HashMap <String,Integer> scenariosRank){
                //Building linkMatrix
                // the linkMatrix represents the interconnections in a matrix form
                // if linkMatrix[i][j]=1
                //HashMap <String,Integer> scenariosRank (specify a scenario it tells your
the rank)
                //HashMap <Integer,String> rankScenarios (specify a rank it tells your the
scenario)
                int nLink = transitions.size();
                int nNodes = scenariosRank.size();
                int[][]linkMatrix = new int[nNodes][nNodes];
```

```
                for (int lk=0;lk<nLink;lk++){
                        String [] defineLink=transitions.get(lk);
                        if
((scenariosRank.containsKey(defineLink[0]))&&(scenariosRank.containsKey(defineLink[1]))){
                                Integer x = scenariosRank.get(defineLink[0]);Integer y =
scenariosRank.get(defineLink[1]);
                                linkMatrix[x][y]=1;
                        }else{

        if(scenariosRank.containsKey(defineLink[0])){JOptionPane.showMessageDialog(null,"W
arning: \n"+
                                        "Connection   ("+defineLink[0]+">>"+defineLink[1]+")
can not be created because "+defineLink[1]+" does not exist.\n"+
                                                "This connection will be ignored.");
                                }else{JOptionPane.showMessageDialog(null,"Warning: \n"+
                                                "Connection
("+defineLink[0]+">>"+defineLink[1]+") can not be created because "+defineLink[0]+" does
not exist.\n"+
                                "This connection will be ignored.");
                }}}
                return linkMatrix;
        }
        public  static  ArrayList  <String[]>  ScenariosScheduler(ArrayList  <String[]>
seqArray,HashMap <String,String[]> scenariosMap){
                ArrayList <String[]> scenariosSchedule=new ArrayList <String[]> ();
                // The element in the scenariosSchedule list a series of sequences. The
information contained in the String[] define:
                // [0]- Name of the scenario
                // [1]- The configuration used for the scenario
                // [2]- The criticality level used to defined the requirements
                // [3]- The mission phase
                // [4]- Reinitiation variable (0 if normal scenario, 1 if placeholder for
restart)
                int nSeq = seqArray.size();// Number of sequences detected
                for (int n=0; n<nSeq; n++){
                        String[] sequence=seqArray.get(n);
                        int nSce = sequence.length;
                        for (int sc=0;sc<nSce;sc++){
                                String [] scDesc={"","","","",""
};
                                if (sc==0){// This is the "start" or initiation node (do
nothing)
                                }else if (sc==nSce-1){
                                        if (n<nSeq-1){
                                        // The is the "end" or FinalNode (here we put set a
mark to specify that the next state restarts a new sequence)
                                        scDesc[4]="1";
                                        scenariosSchedule.add(scDesc);//Register    scenario
name
                                        }else{
                                        }//This is the last scenario no need to register a
restart scenarios since no sequence are following.
                                }else{ // Normal scenario
                                        String scenario = sequence[sc];
                                        String[]    scD=scenariosMap.get(scenario);//    Get
description of scenario
                                        // Add the deactivated reinitiation variable
                                        scDesc[0]=scenario;
                                        for(intk=0;k<3;k++){
                                                scDesc[k+1]=scD[k];} scDesc[4]="0";

                                        scenariosSchedule.add(scDesc);//Register    scenario
name
                                }//Done copying the description of the scenario
                        }//Done copying the sequence
                }//All sequences were copied

                return scenariosSchedule;
        }
}
```

Appendix J

Framework User Guides

In order to facilitate the reproduction and utilization of the framework implemented in support of this dissertation, a user guide was implemented. This user guide is represented in the form of snapshots of the powerpoint slides. This user guide is organized in three parts.

The first presents the installation and configuration process associated with the installation of the different environments constituting this framework. The second user guide explains how new subsystem model can be implemented (both in the SysML environment and the Analysis Server library). The third user guide presents the process to define an architecture concept in SysML.

The author would like to thank Jonathan Herault and Charles Nespoulous[6] for reviewing these guides.

[6] Charles and Jonathan were Graduate Researcher Assistant at the Aerospace Systems Design Laboratory in Spring 2010.

J.1 Installation Guide

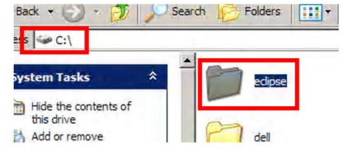
User guide for setting up the architecture builder

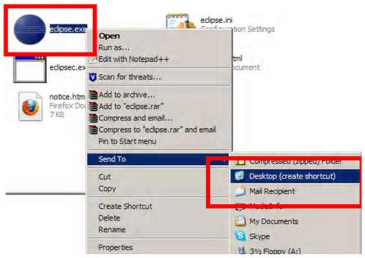
Installing MagicDraw

- Download the latest version of Magic Draw available at: www.magicdraw.com/
- Make sure that you have the following licenses:
  - MagicDraw_16_6_Enterprise_Site_key.txt
  - MagicDraw_SYSML_16_6_Academic_Site_key.txt
- Proceed by following the instructions indicated by the installation software

Installing Eclipse

1. Download eclipse or use the version provided (on the key)
2. The folder downloaded must pasted in C:\
3. Make a link of the executable on the desktop



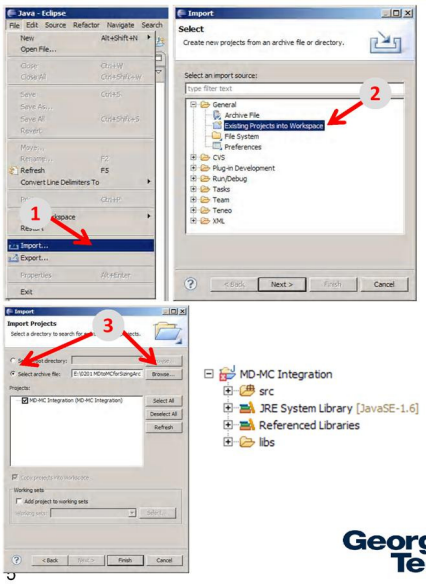


First session with Eclipse

1. Double click the link to the executable
2. Use the default workspace location
3. Go to the workbench (this will be your home)

Import the workspace

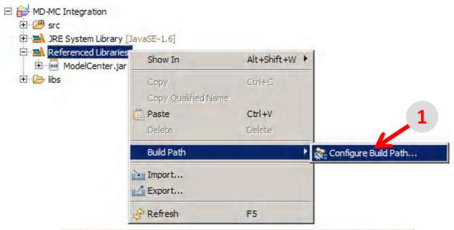
1. Import the workspace (under the file menu)
2. Select “Existing project into Workspace” (click next)
3. Select archive file then browse to the zip file in the USB key

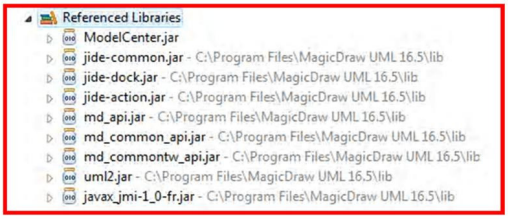


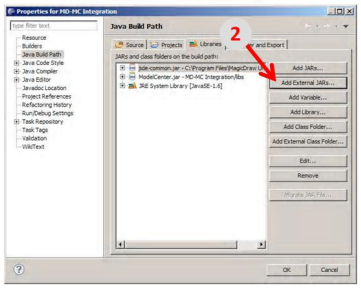
Fix references in eclipse libraries

1. Access the libraries by right clicking on “reference libraries”

   Delete all libraries with a red X
2. Add external JARs using the paths indicated bellow



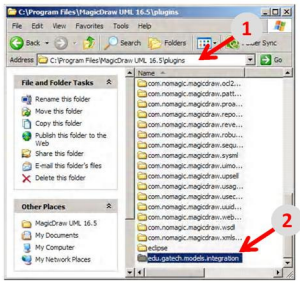




Create JAR folder hosting the builder

1. From explorer navigate to the MagicDraw folder /plugins
2. Copy paste the folder edu.gatech.models.integration

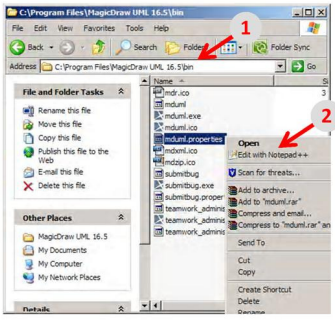
   (Note this folder is located on the key)

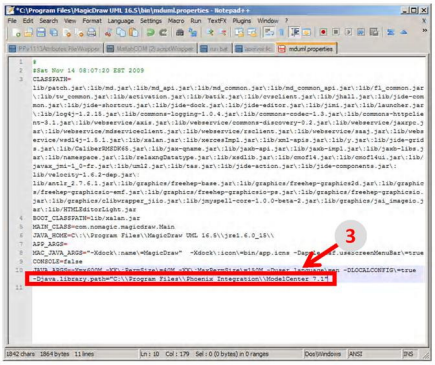


Fix Magic Draw reference to Model Center

1. From explorer navigate to the MagicDraw folder /bin
2. Edit the file named: mduml.properties
3. On the last line of the file paste the following string: -Djava.library.path="C:\\Program Files\\Phoenix Integration\\ModelCenter 7.1"

   Note on step 3: The path indicated above must correspond to the folder containing ModelCenter



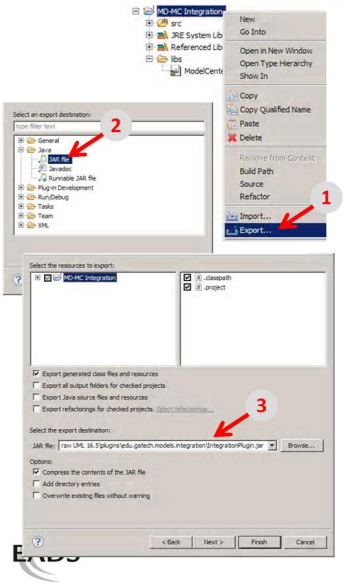


Warning:

In step three do NOT create a new line (copy the string at the end of the existing line with a space)

Sometimes the double quote at the end does not work. After perfoming step 3 restart magic draw. If it doesn not work there is a chance the double quote was corrupted. Just erase it and replace by a new double quote "

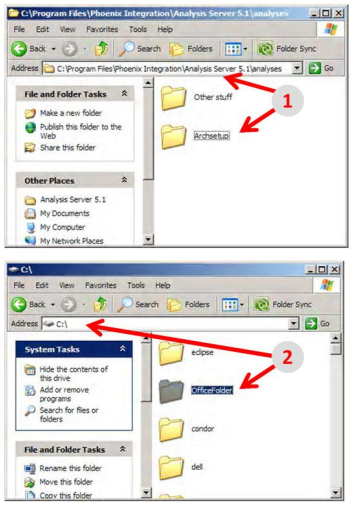
Export architecture builder to JAR folder



1. Right click on Java project and select Export
2. Export as JAR file
3. Select the IntegrationPlugin.jar file located in the MagicDraw plugins folder: plugins\edu.gatech.models.integration\IntegrationPlugin.jar

Installation of model folders

1. Install subsystem model folders: Access the analysis server directory and paste Archsetup in the analyses folder
2. Install architecture model folders: Under C:\

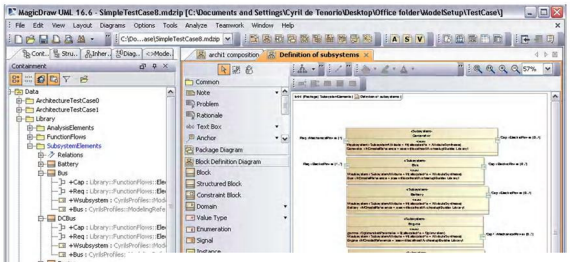


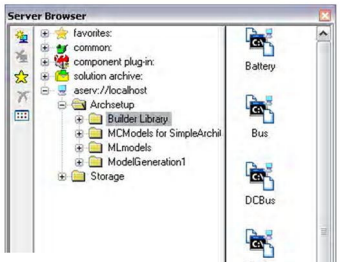
J.2 Implementation of New Subsystems

Implementation of New Subsystem Types

By Cyril de Tenorio

Organization of knowledge





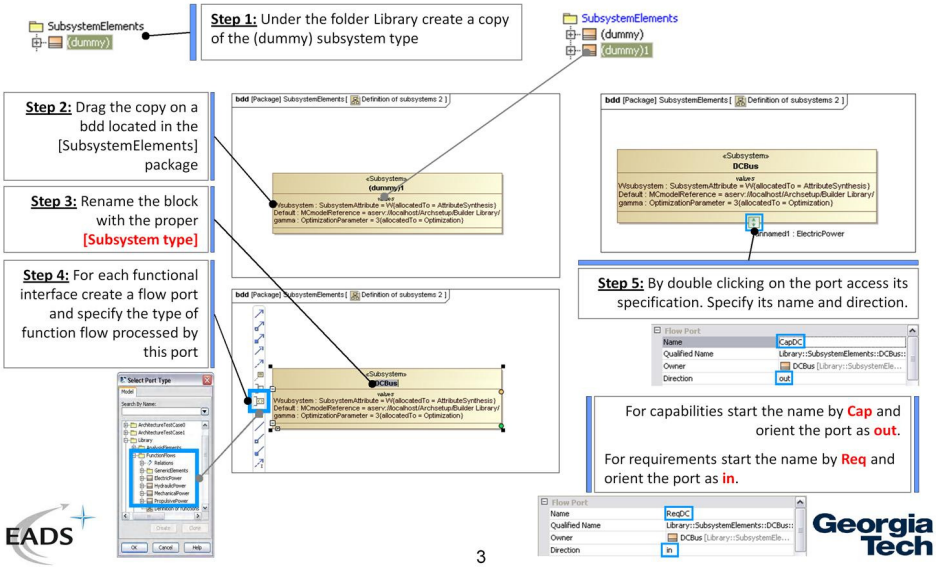
- The types defined in Magicdraw correspond to a model stored in the Analysis Server
- We must ensure that the subsystem type and model correspond to each other in terms of interfaces
  - Functional interfaces
  - Attributes
- This guide will first describe
  - the definition of the subsystem types in MagicDraw
  - The establishment of the matlab model elements in model center

EADS

Georgia Tech

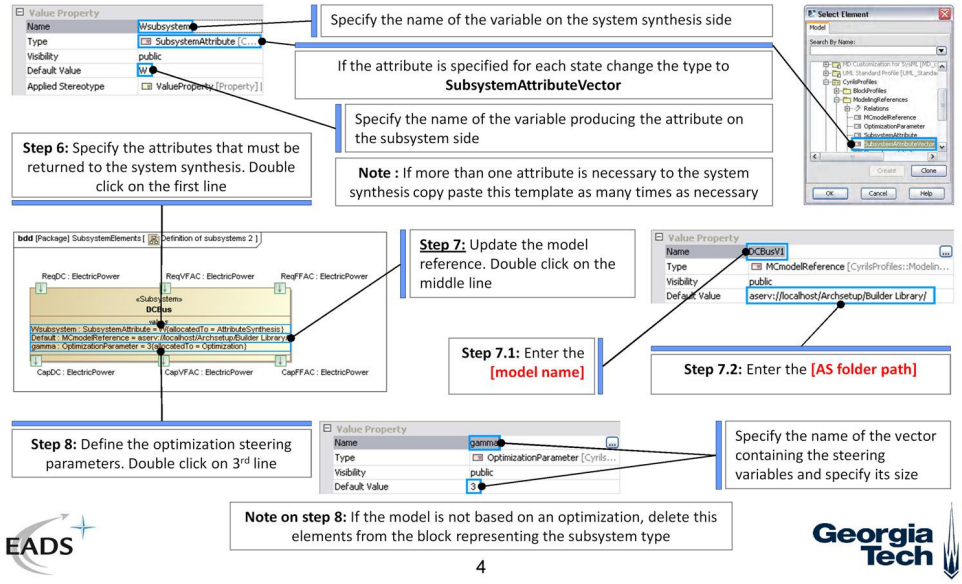
2

Constructing the subsystem type



3

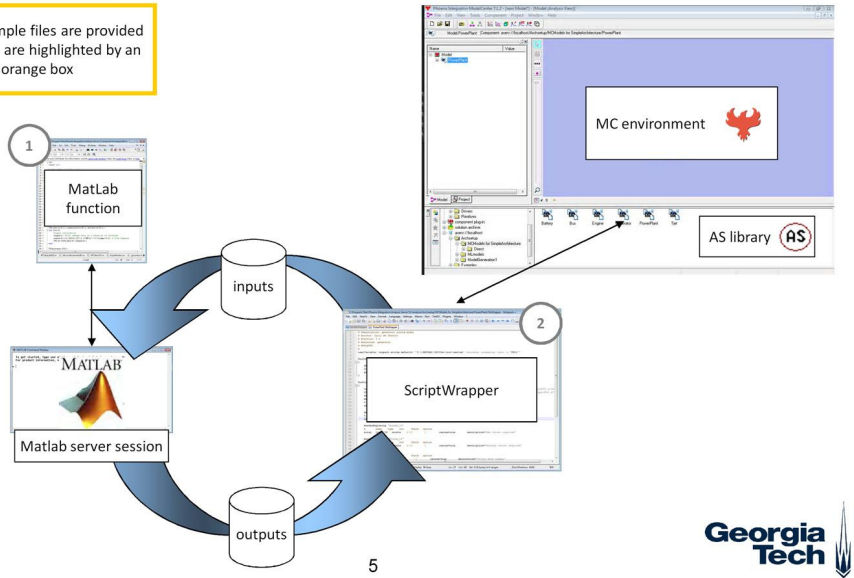
Constructing the subsystem type



4

Wrapping Matlab scripts in AS (with VBScript wrapper)

Note: Example files are provided these files are highlighted by an orange box



EADS

Georgia Tech

Step 1: Creation of the Matlab ScriptWrapper

```
% Step 1: Give your MC model a name
ModelName = 'TestingGen';

% Step 2: Specify the ML model name (make sure your function is on the ML path)
MLfctName = 'GasTurbineF';

% Step 3: Define the inputs in your function
% 3.1: the variables have to be listed in the same order as they show in
%       your function declaration.
% 3.2: for each specify if the variable is a scalar or a vector
Inputs = {'T41',scalar,1800
    'LoadM',vector,[22,10]
    'LoadN',vector,[20,8]
    'RPMnom',vector,[1000,1000]
    'SpRat',vector,[1.8,1.8]
    'Ta',vector,[30,10]
    'Pa',vector,[10e5,10e5]
    'Ma',vector,[0.2,0.3]};

% Step 4: List the outputs (same as step 3)
Outputs = {'Weight',scalar
    'mf',vector};

% Step 5: Specify the folder in which the scriptWrapper must be stored
filePath='C:\Program Files\Phoenix Integration\Analysis Server 5.1\analyses\MatlabExe\';
```

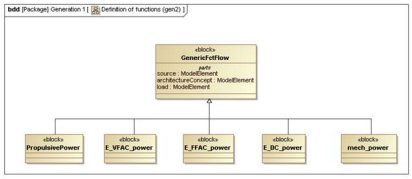
Follow directives indicated in the beginning of the code

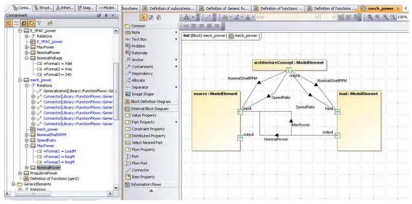
TBD

EADS

Georgia Tech

Creating a new function





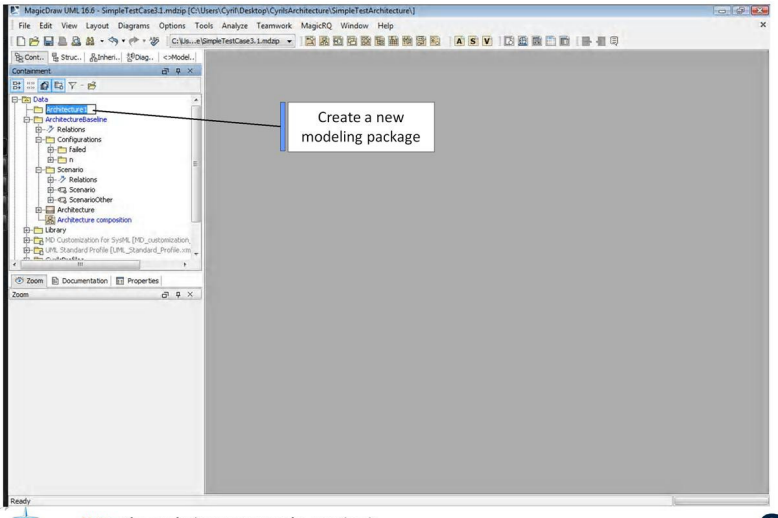
- Create a new block
- Make it a specialization of the “GenericFctFlow” block
- Name to the function by giving a name to its block
- Create an ibd in the block representing the function flow
- Make sure you show all the parts and ports
- Delete the input port on the “architectureConcept” part
- For each connection:
  - Make the link
  - Create the flow
  - For each flow make a block
    - The name of the block describes the information conveyed by the connection
    - The block has 3 value properties
      - Format1: Name of the variable on the source side
      - Format2: Name of the variable on the load side
      - Format 3: Name of the variable provided by the mission (if the function flow is assigned on a Boundary Function) or default value defined in the architecture

J.3 Definition of the Architecture Concept

User guide for describing an architecture (while building its sizing model)

By Cyril de Tenorio

Step 0: Creation of the modeling package hosting the description of architecture concept



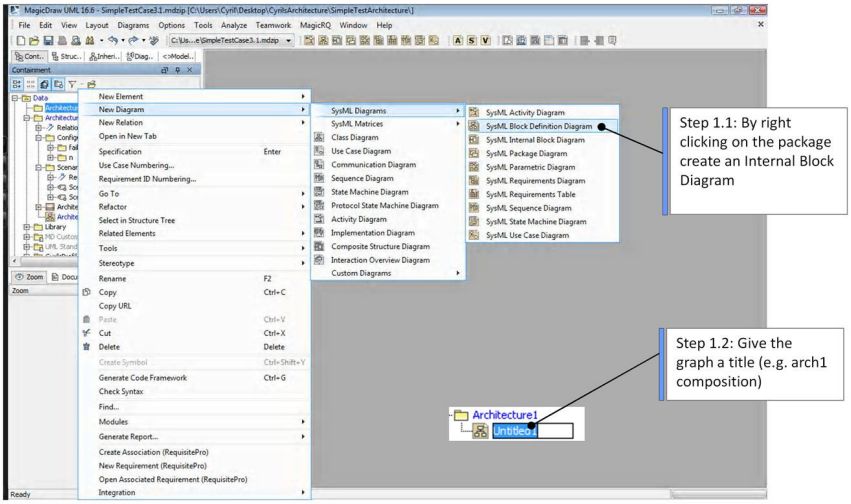
EADS

Note: The analysis server must be running in the background when ever the builder is used

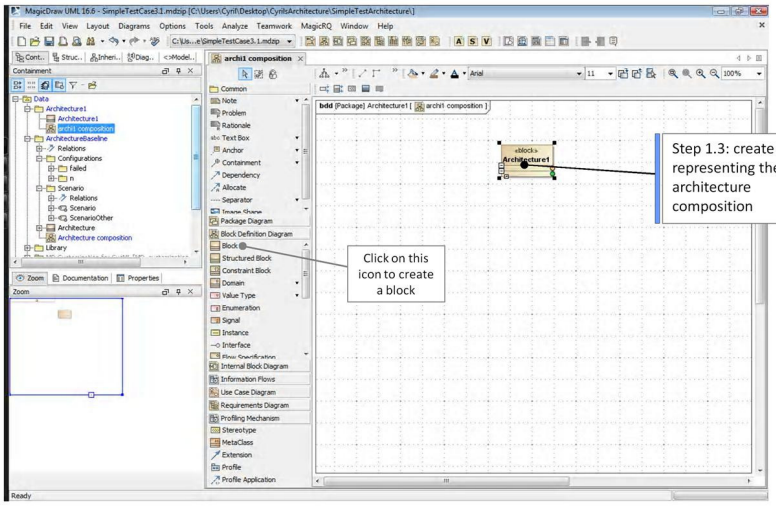
2

Georgia Tech

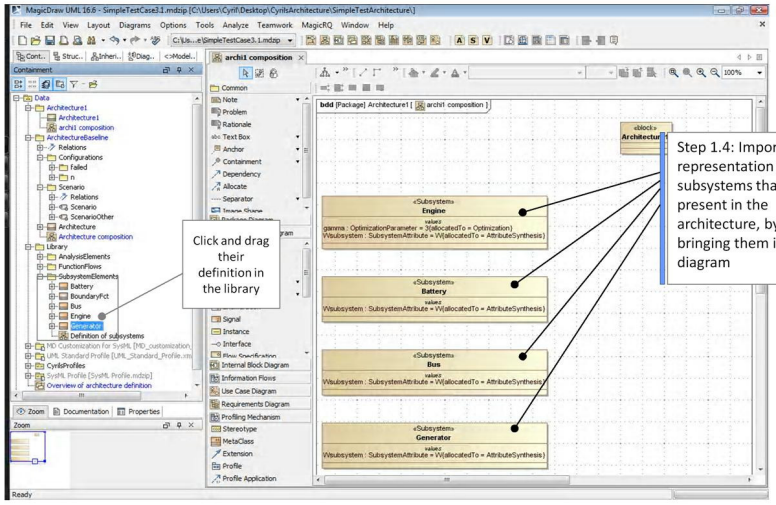
Step 1: Create architecture composition



Step1: Create architecture composition



Step1: Create architecture composition

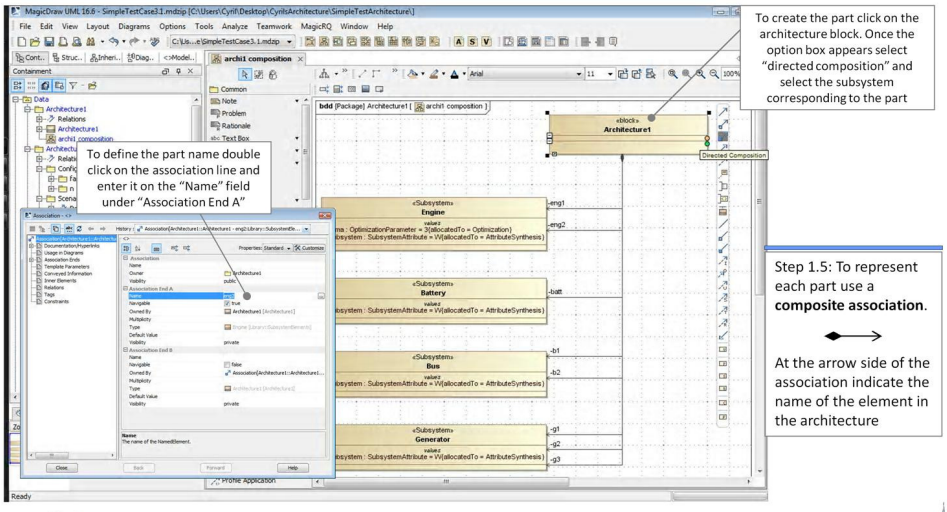


EADS

Georgia Tech

5

Step1: Create architecture composition

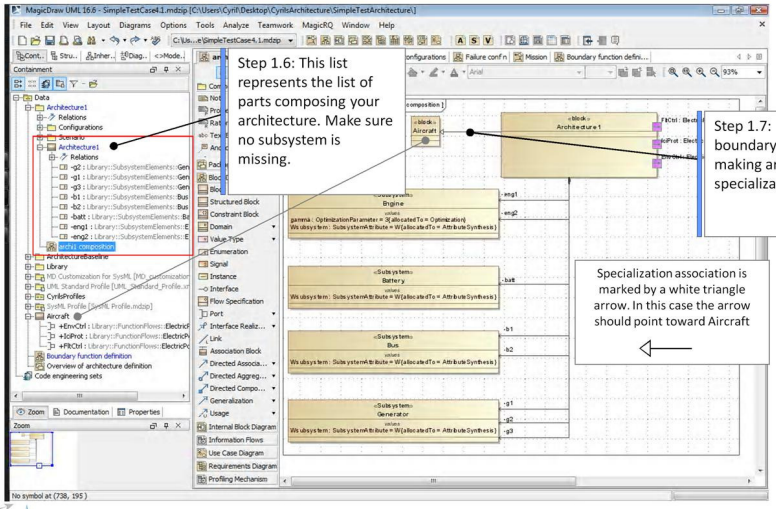


EADS

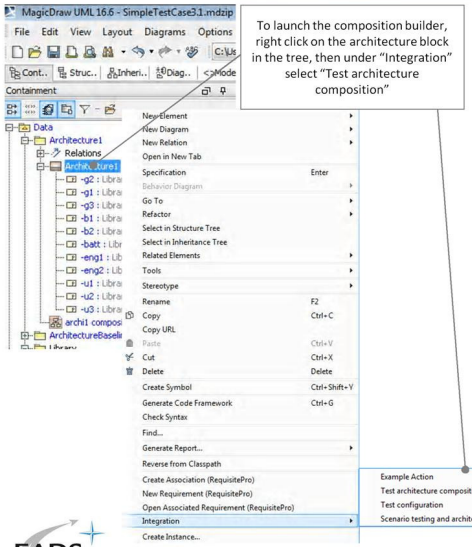
Georgia Tech

6

Step1: Create architecture composition

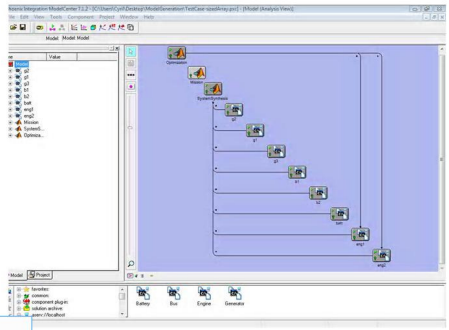


Step1: Create architecture composition



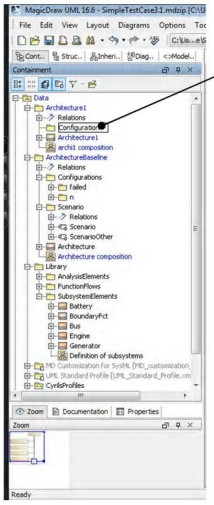
Step 1.8 (optional but recommended):

At this point the composition section of the model building routine should work. Run it and inspect that all subsystems are represented by specific instances of the analysis



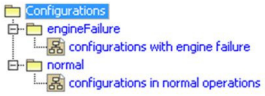
Note: The analysis server must be running in the background when ever the builder is used

Step 2: Define the configuration



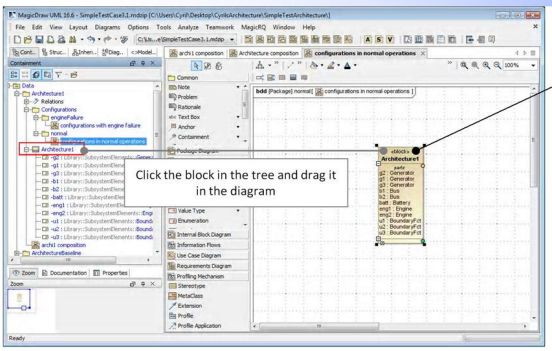
Step 2.1: Create a package containing the configurations. If different failure configurations are considered, I recommend to make a different package for each.

Step 2.2: Under each package create and name an block definition diagram. (similar procedure as step 1.1)



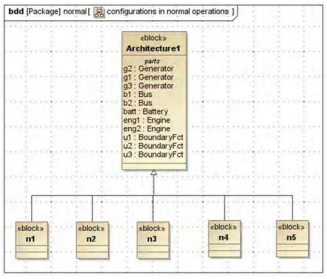
Note: The next procedures have to be repeated for each configuration packages

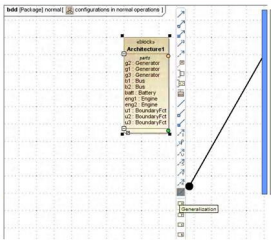
Step 2: Define the configuration



Step 2.1: In the newly created bdd, import the architecture block.

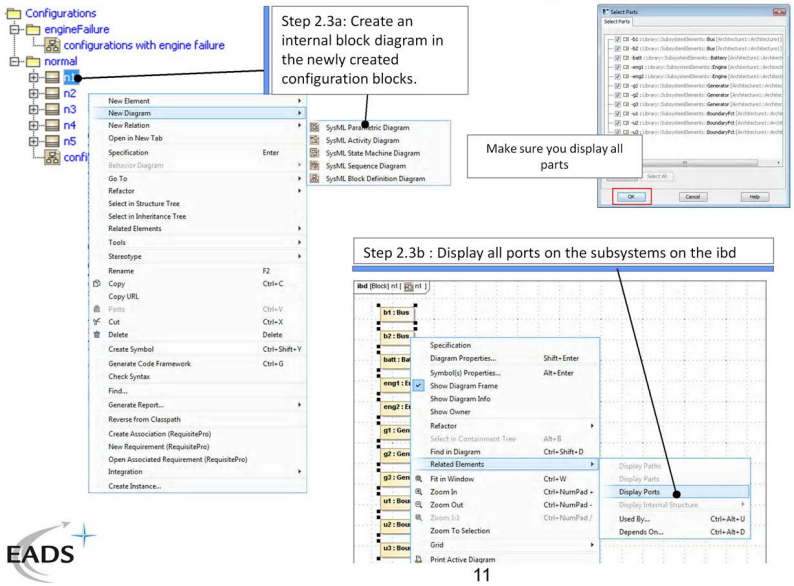
Click the block in the tree and drag it in the diagram





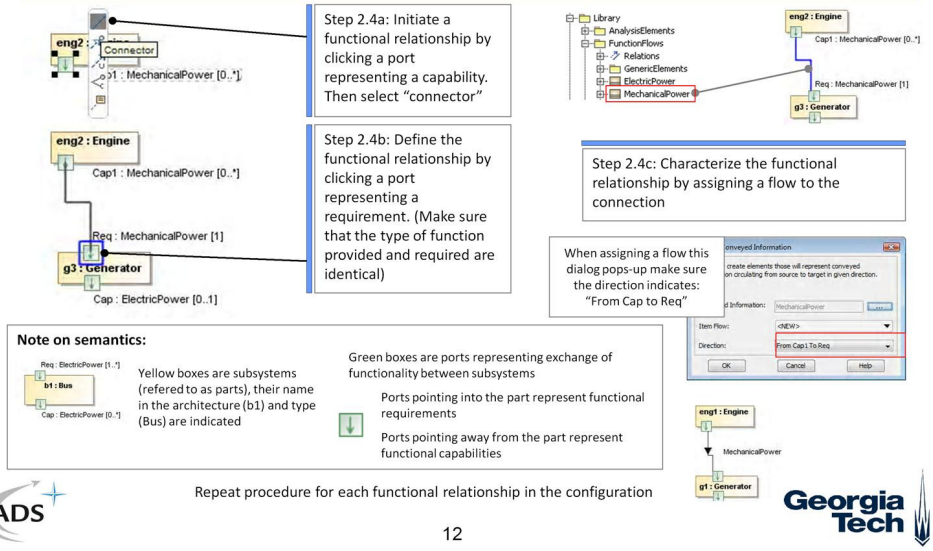
Step 2.2: Define you configuration by creating blocks defined as "specialization of" architecture. Give it a name (make sure it is not ambiguous with other configuration names)

Step 2: Define the configuration



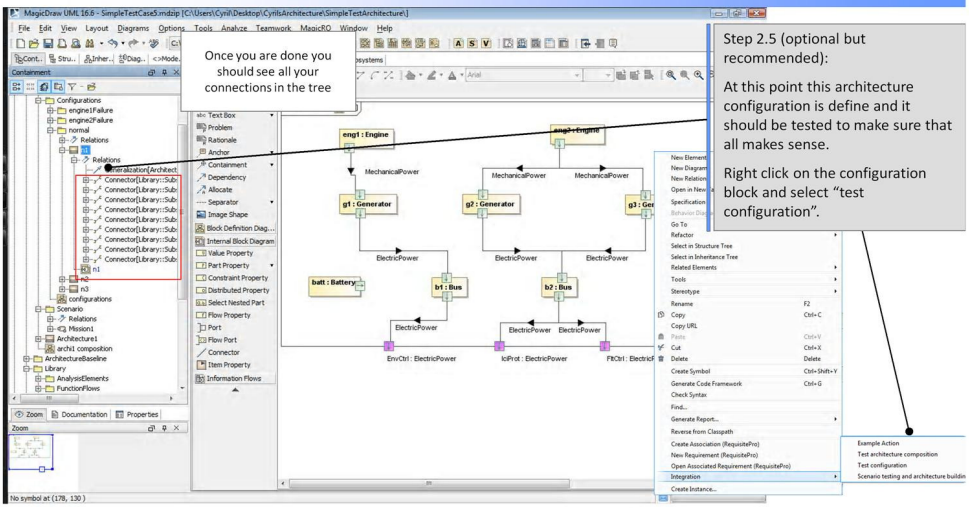
Step 2: Define the configuration

Step 2.4: Create the functional relationships active in this configuration



Repeat procedure for each functional relationship in the configuration

Step 2: Define the configuration



Note: You can run this only if the composition has been run once before (see step 1.8)

Step 2: Define the configuration

Often configuration are similar and only differ by one or few different connections. In order to accelerate the definition process the architect can copy paste a configuration and adapt to the new configuration.

Procedure:

Copy the block representing the configuration.

Paste the block in the package appropriate.

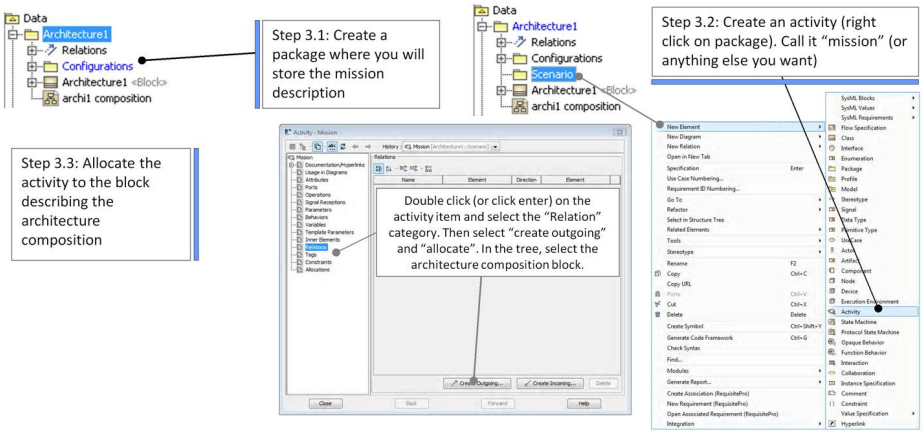
Give the appropriate name to the new configuration

For reasons that escape me, copy/pasting the configuration does not reproduce the functional flow assignments. Therefore you will have to reproduce step 2.4c for all functional relationships in the configuration.

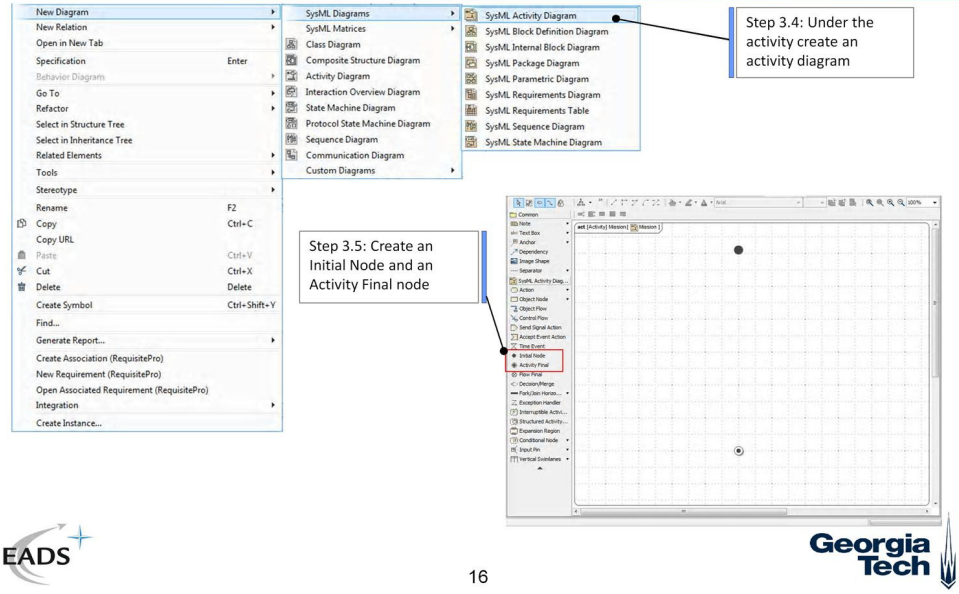
In order to avoid any bad surprise, validate your configuration

Note: The analysis server must be running in the background when ever the builder is used

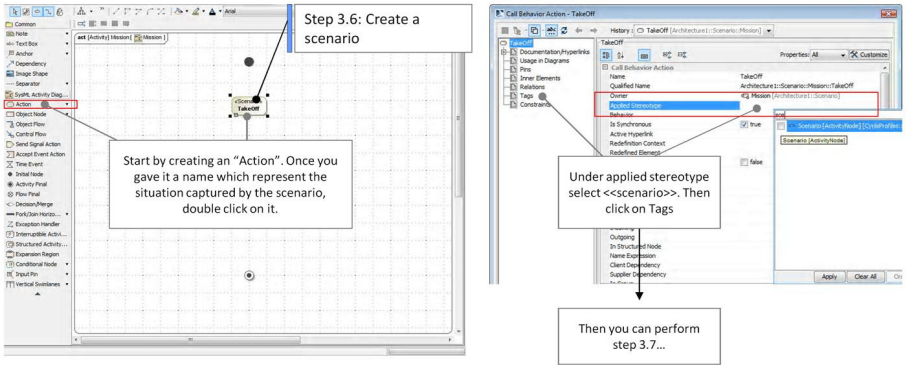
Step 3: Definition of the mission scenarios and sequences



Step 3: Definition of the mission scenarios and sequences

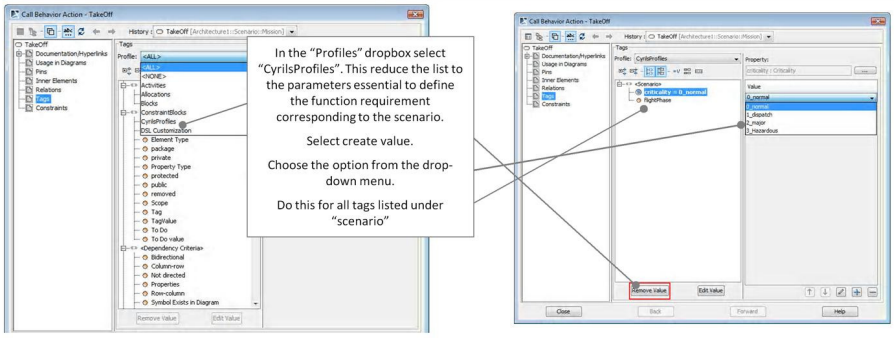


Step 3: Definition of the mission scenarios and sequences



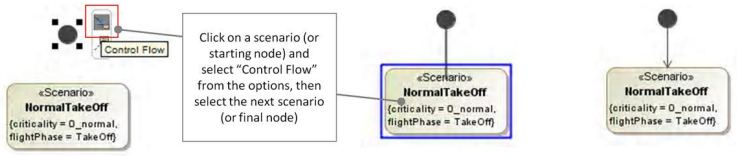
Step 3: Definition of the mission scenarios and sequences

Step 3.7: Define the scenario by defining its mission parameters



Step 3: Definition of the mission scenarios and sequences

Step 3.8: Create the transitions



Suggested name convention for transition:

If the transition between two scenarios implies

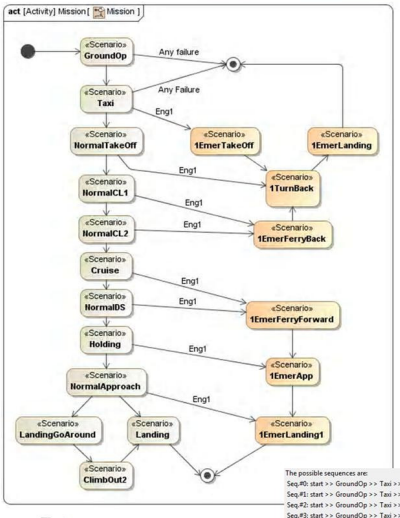
- no new failure event (e.g. going from a normal take-off to a normal climb) no name is used to label the transition
- a new failure occurred (e.g. normal cruise to emergency descent with failed engine) a label is recommended

NB: These labels are not considered by the model but improve the clarity of the graph

EADS

Georgia Tech

Step 3: Definition of the mission scenarios and sequences



Important feature of the activity diagram:

1. **No Cycles:** The graph may not include cycle cycles. As the sequence will be defined, the builder will not know when to exit the cycle and will return an error message.
2. **Single initial and ending nodes:** There must be precisely one instance of the initial node and precisely one instance of the final node. Make sure you check the containment tree to verify that only one instance is present in the description.

Example of a mission sequence:

This diagram represent an example of an activity diagram.

Cycles: to avoid the presence of cycle, I encourage to use different names for similar scenarios occurring several times in the mission. In this example, the normal climb-out is represented twice in the mission, if they were given the same name the builder could think that they correspond to the same situation and that the pilot after going around may decide to continue to climb to cruise sequence (which obviously does not make sense).

Initial/Ending nodes: These terminal nodes can be represented several times in the diagram, but the architect must make sure that each instance represent the same instance in the tree.

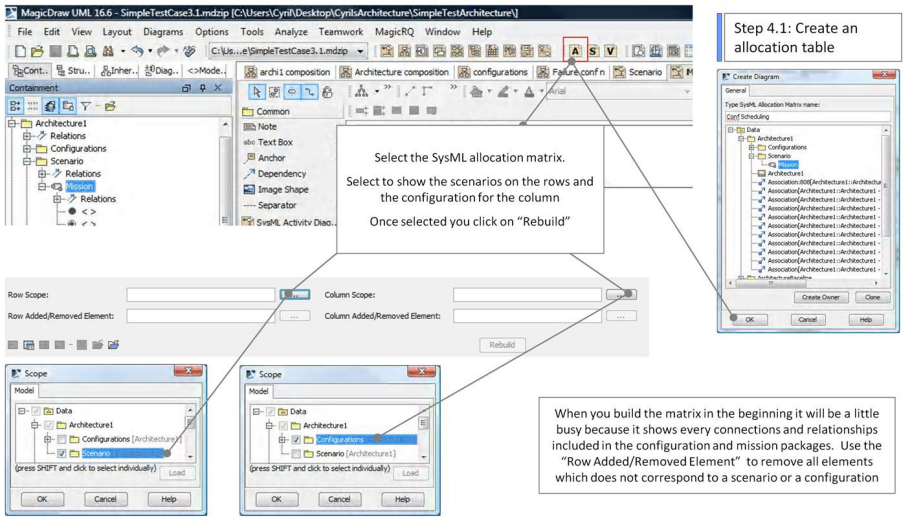
Colors: It is possible to adapt the color of the scenario of the graph by selecting activities nodes and clicking (alt+enter).

The possible sequences are:

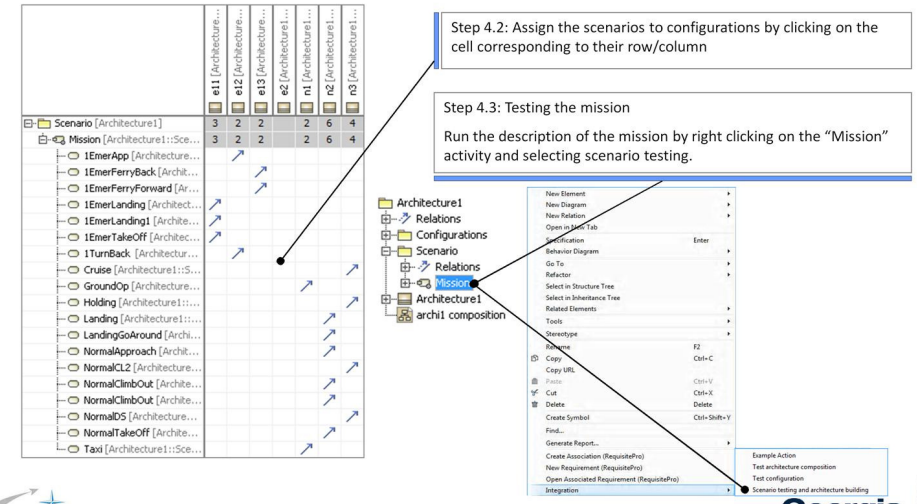
Seq.#0: start >> GroundOp >> Taxi >> NormalTakeOff >> NormalClimbOut1 >> NormalCL2 >> Cruise >> NormalDS >> Holding >> NormalApproach >> LandingGoAround >> NormalClimbOut2 >> Landing >> end >>
Seq.#1: start >> GroundOp >> Taxi >> 1EmerTakeOff >> 1TurnBack >> 1EmerLanding >> end >>
Seq.#2: start >> GroundOp >> Taxi >> NormalTakeOff >> 1TurnBack >> 1EmerLanding >> end >>
Seq.#3: start >> GroundOp >> Taxi >> NormalTakeOff >> NormalClimbOut1 >> 1EmerFerryBack >> 1TurnBack >> 1EmerLanding >> end >>
Seq.#4: start >> GroundOp >> Taxi >> NormalTakeOff >> NormalClimbOut1 >> NormalCL2 >> 1EmerFerryBack >> 1TurnBack >> 1EmerLanding >> end >>
Seq.#5: start >> GroundOp >> Taxi >> NormalTakeOff >> NormalClimbOut1 >> NormalCL2 >> Cruise >> 1EmerFerryForward >> 1EmerApp >> 1EmerLanding1 >> end >>
Seq.#6: start >> GroundOp >> Taxi >> NormalTakeOff >> NormalClimbOut1 >> NormalCL2 >> Cruise >> NormalDS >> 1EmerFerryForward >> 1EmerApp >> 1EmerLanding1 >> end >>
Seq.#7: start >> GroundOp >> Taxi >> NormalTakeOff >> NormalClimbOut1 >> NormalCL2 >> Cruise >> NormalDS >> Holding >> 1EmerApp >> 1EmerLanding1 >> end >>
Seq.#8: start >> GroundOp >> Taxi >> NormalTakeOff >> NormalClimbOut1 >> NormalCL2 >> Cruise >> NormalDS >> Holding >> NormalApproach >> 1EmerLanding1 >> end >>
Seq.#9: start >> GroundOp >> Taxi >> NormalTakeOff >> NormalClimbOut1 >> NormalCL2 >> Cruise >> NormalDS >> Holding >> NormalApproach >> Landing >> end >>

EADS

Step 4: Scheduling configurations

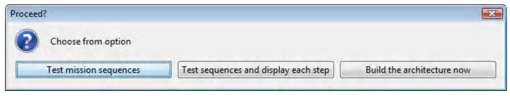


Step 4: Scheduling configurations



Step 4: Scheduling configurations

Step 4.3: Verifying the mission sequences



Two checking options are available:

Test mission sequences: This will run import the diagram and build the sequence. In this setting none of the steps are shown, but it will return an angry message to the screen if something was not defined properly

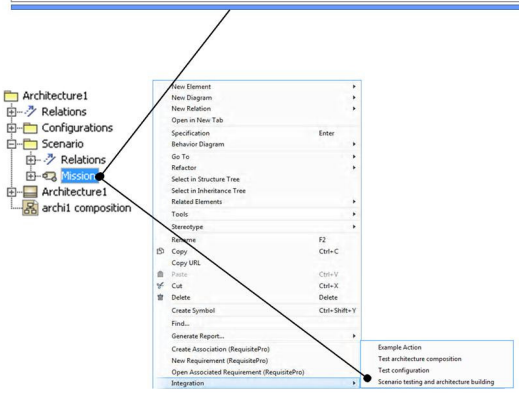
Test mission sequence and display each step: This option is recommended as it show everything that the builder is importing. For each scenario it will present its associated information. It is a good way to verify that no elements in the definition of the scenario was neglected or improperly defined.
Then it will display the collection of transitions present in the activity diagram. Visually inspect this list to unsure that all connections are complete and realistic. It will display the list of all scenario detected and incorportated. Make sure that all scenarios are present and that there are no redundancy.
Finally you will have the oportunity to check the construction of the scenario sequences. If you see that the sequence are stuck on a loop verify your graph.

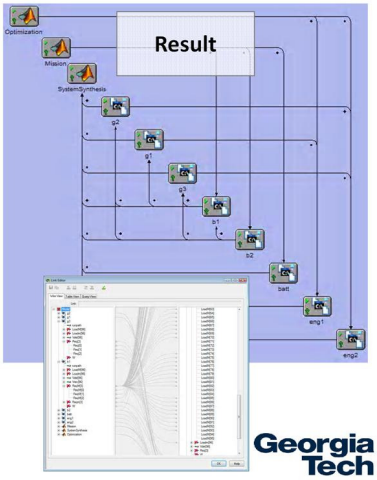
It is common that by improperly copy pasting scenario in the activity diagram, some cyclical sequence arise. This is due to the fact that transition are sometimes copied along with the scenario (even if they are not displayed on the graph). Therefore verify that the transitions displayed by the builder and make sure they are real. The sequence builder to the right is a practical mean to check them.

Note: The analysis server must be running in the background when ever the builder is used

Step 5: Building the architecture

Step 5: Don't worry if every thing before this point has work this will work. Build by right clicking on the "Mission" activity and selecting scenario testing.





Note: The analysis server must be running in the background when ever the builder is used

Appendix K

System Synthesis Block

```
% variable: Scenario string[] input
% variable: Criticality string[] input
% variable: FlightPhase string[] input
% variable: Cruisek double input
% variable: DT double[] input
% variable: TOGW double input
% variable: BasePowSysW double input
% variable: BaseFuel double input
% variable: Wfuel double output
% variable: FuelFlows double[] input
% variable: TotFFlow double[] output
% variable: Wsubsystem double[] input
% variable: Wtot double output
% variable: OpRange double output
% variable: Duration double output

%% Synthesis %%
%%%%%%%%%%%%%%%
% Estimating the overal weight
Wtot = sum(Wsubsystem);

% Estimation of fuel on board
Overweight=Wtot-BasePowSysW;
Wfuel=BaseFuel-Overweight;
K=length(DT);
% Detecting the end of the mission
k=1;
while DT(k)>0&&k<K
    k=k+1;
end
Endk=k;
clear k

% Fuel flow estimation
TotFFlow = zeros(1,Endk);
for k = 1: Endk
    TotFFlow(k)=sum(FuelFlows(:,k));
end
FFcruise=TotFFlow(Cruisek);

% Fuel burn estimation
TotFB = zeros(1,Endk);
for k = 2: Endk
    if DT(k-1)<0
        TotFB(k)=0;
    else
        TotFB(k)=TotFB(k-1)+TotFFlow(k-1)*DT(k-1)*60;
    end
end
FuelReserve= Wfuel-TotFB;

FBclimb = TotFB(Cruisek-1);                 % Fuel burn to climb
FBland  = TotFB(Endk)-TotFB(Cruisek);       % Fuel burn to land
FBCruise = Wfuel-FBland-FBclimb;            % Fuel available to cruise
Duration = FBCruise/FFcruise;               % Allowable duration for cruise
OpRange=Duration*800/60;                       % Operational range

plot([0:Endk-1],TotFB(1:Endk))
```

References

1. European Committee for Standardization, *Functional and Technical Specifications*. 2005.

2. MacPherson, A. and V. Vanchan, *The Outsourcing of Industrial Design Services by Large US Manufacturing Companies.* International Regional Science Review, 2009: p. 0160017608330266.

3. Ibsen, A.Z., *The politics of airplane production: The emergence of two technological frames in the competition between Boeing and Airbus.* Technology in Society, 2009. **31**(4): p. 342-349.

4. Beath, J., Y. Katsoulacos, and D. Ulph, *Strategic R & D Policy.* The Economic Journal, 1989. **99**(395): p. 74-83.

5. Giovannetti, E., *Perpetual leapfrogging in Bertrand Duopoly*, Dipartimento di Economia Pubblica, Universita' degli studi di Roma "La Sapienza".

6. MacPherson, A. and D. Pritchard, *Boeing's Diffusion of Commercial Aircraft Technology to Japan: Surrendering the U.S. Industry for Foreign Financial Support.* Journal of Labor Research, 2007. **28**(3): p. 552-566.

7. Pritchard, D. and A. MacPherson, *Outsourcing US commercial aircraft technology and innovation: implications for the industry's long term design and build capability.* Canada – United State Trade Center - Occasional paper, 2004.

8. Anderson, J.D., *Aircraft performance and design*. 1999: W.C. Brown Pub. Co.

9. Mavris, D., *Aircraft design - class material*. 2006, Georgia Institute of Technology: Atlanta, Georgia.

10. Paredis, C., *Modeling and Simulation, class material*. 2008, Georgia Institute of Technology: Atlanta Georgia.

11. Mavris, D., *Advanced Design Methods I, class material*. 2006, Georgia Institute of Technology: Atlanta, Georgia.

12. Womack, J.P., D.T. Jones, and D. Roos, *The machine that changed the world*. 1990, New York: Harper Perennial.

13. Nicolai, L.M., *Fundamentals of aircraft design*. 1975, San Jose, California: Mets Inc.

14. *FACTBOX-Sizing up the Airbus A380 double-decker*, in *Reuters*. 2007.

15. Gandy, A. and M. Lepesant, *A350. La nouvelle organisation industrielle d'Airbus*, in *Le journal des entreprises*. 2009.

16. de Chazelles, P., et al. *Cadre d'application de l'Ingenierie Systeme pour la conception d'un avion commercial*.

17. Cant, T., J. McCarthy, and R. Stanley, *Tools for requirements management: a comparison of Telelogic DOORS and the HiVe*. 2006, Information networks Division Defense Science and Technology Organisation: Edinburgh, South Australia.

18. Telelogic. *Telelogic Customers, http://www.telelogic.com/Customers/*. 2008.

19. *Massive 787 Electrical System Pressurizes Cabin*, in *Aviation week & space technology*. 2005. p. 47.

20. Okabe, N. *L'aéroport d'Osaka*. in *Université de touts les savoirs*. 2004.

21. Rechtin, E., *The art of architecting.* IEEE Spectrum, 1992(October 1992): p. 66.

22. Liscouet-Hanke, S., *A Model-Based Methodology for Integrated Preliminary Sizing and Analysis of Aircraft Power System Architectures*, in *Institut National des Sciences Appliquee - Laboratoire de Genie Mecanique de Toulouse*. 2008, Universitee Paul Sabatier: Toulouse.

23. Faleiro, L., *"Power by Wire" Aircraft -Vision or Myth?"*, in *SAE Power Systems Conference*. 2006: New Orleans, Louisiana.

24. Guyot, J.C., *Discussion on ATV power architecture developments*. 2008: Les Mureaux.

25. Mattingly, J.D., *Aircraft Engine Design*, ed. A.E. Series. 2002.

26. Drela, M., *Transport Aircraft Optimization*, in *AE Fall 2009 Seminar Series*. 2009, Georgia Institute of Technology: Atlanta.

27. Akiyama, K., *Function Analysis: Systematic Improvement of Quality and Performance*. 1989, Tokyo, Japan: Productivity Press.

28. Huo, B., et al., *Optimized Aircraft Power Architecture: Integrated Product and Process Development (IPPD) - final project*. 2006, Georgia Institute of Technology: Atlanta, Georgia.

29. Mavris, D., C. de Tenorio, and M. Armstrong, *Methodology for Aircraft System Architecture Definition*, in *46th AIAA Aerospace Science and Sciences Meeting and Exhibit*. 2008: Reno, Nevada.

30. Bytheway, C.W., *FAST Creativity and Innovation*. 2006: J.Ross publishing.

31. Sage, A.P. and W.B. Rouse, *Handbook of systems engineering and management*. 1999, New York: Wiley & Sons.

32. Sage, A.P. and J.E. Armstrong, *Introduction to systems engineering*. 2000, New York: Wiley and sons.

33. Armstrong, M., C. de Tenorio, and D. Mavris, *Function based architecture design space definition and exploration*, in *26th International Congress of the Aeronautical Sciences*. 2008: Anchorage, Alaska.

34. Romli, F., *Aircraft System - Subsystem breakdown (ATA 100)*. 2005, Georgia Institute of Technology.

35. Zwicky, F., *Morphological analysis and construction*, ed. W. Inter-science. 1948, New York.

36. Engler, W.O., P.T. Biltgen, and M. Dimitri, *Concept Selection Using an Interactive Reconfigurable Matrix of Alternatives (IRMA)*, in *45th AIAA Aerospace Sciences Meeting and Exhibit*. 2007: Reno, Nevada.

37. Pilone, D. and N. Pitman, *UML in a Nutshell*. 2005: O'Reilly.

38. OMG, *OMG Systems Modeling Language (OMG SysML^{tm}), v1.1*. 2008.

39. Paredis, C. and L. McGinnis, *System Modeling with SysML*, in *ISyE 8813*. 2008, Georgia Institute of Technology: Atlanta.

40. Friedenthal, S., A. Moore, and R. Steiner, *A practical guide to SysML*. The Systems Modeling Language. 2008: Morgan Kaufmann OMG Press.

41. Sage, A.P., *Methodology for large-scale systems*. 1977: McGraw-Hill Book Company.

42. Cole, B., *An architectural evolution framework based on Pacelab*, in *Pace Days*. 2008: Atlanta, Georgia.

43. Cole, B.F., *An evolutionary Method for Synthesizing Technological Planning and Architectural Advance*, in *Guggenheim School of Aerospace Engineering*. 2009, Georgia Institute of Technology: Atlanta.

44. Mabu, S., K. Hirasawa, and J. Hu, *A graph-based evolutionnary Algorithm: Genetic Network Programming (GNP) and its extention using reinforcement learning*. Evolutionary Computation, 2007. **15**(3): p. 369-398.

45. Armstrong, M., *A process for function based architecture definition and modeling*, in *School of Aerospace Engineering*. 2008, Georgia Institute of Technology.

46. de Tenorio, C., et al., *Methodology for Aircraft System Architecture Sizing*, in *International Congress of Aeronautical Sciences*. 2008: Anchorage, Alaska.

47. de Tenorio, C., M. Armstrong, and D. Mavris, *Architecture Subsystem Sizing and Coordinated Optimization Methods*, in *47th AIAA Aerospace Sciences Meeting and Exhibit*. 2009: Orlando, Florida.

48. de Tenorio, C., *Aircraft System Architecture Design Methods*, in *Guggenheim School of Aerospace Engineering*. 2007, Georgia Institute of Technology: Atlanta.

49. Dieter, G.E., *Engineering design: a material and processing approach*. 2000: McGraw-Hill.

50. Clark, J.P., *Advanced Design Methods II, class material*. 2006, Georgia Institute of Technology: Atlanta, Georgia.

51. Kokkolaras, M., et al., *Simulation-based optimal design of heavy trucks by model-based decomposition: An extensive analytical target cascading case study* Int. J. Heavy Vehicle Systems, 2004. **11**(3/4): p. 402-432.

52. Etman, L.F.P., et al., *Coordination specification in distributed optimal design of multilevel systems using the χ language.* Struct. Mulidisc. Optim, 2005. **29**: p. 198-212.

53. Kokkolaras, M., Z.P. Mourelatos, and P.Y. Papalambros, *Design optimization of hierarchically decomposed multilevel systems under uncertainty.* Journal of mechanical design, 2006. **128**: p. 503-508.

54. Tosserams, S., et al., *An augmented Lagrangian relaxation for analytical target cascading using the alternating direction method of multipliers.* Struct. Mulidisc. Optim, 2006. **31**: p. 176-189.

55. Sobieski, I., et al., *Bi-level integrated system synthesis (BLISS) for concurrent and distributed processing*, in *9th AIAA/ISSMO symposium on multidisciplinary analysis and optimization*. 2002: Atlanta,Georgia.

56. Sobieszczanski-Sobieski, J., et al., *Bilevel Integrated System Synthesis for Concurrent and Distributed Processing* AIAA Journal, 2003. **41**(10).

57. Sobieski, I. and I. Kroo, *Aircraft design using collaborative optimization*, in *34th AIAA Aerospace Sciences Meeting and Exhibit*. 1996: Reno, Nevada.

58. de Wit, A.J. and F. vanKeulen, *Numerical comparison of Multi-level optimization techniques*, in *48th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Material Conference*. 2007: Honolulu, Hawaii.

59. Dassault Systèmes. *Isight, http://www.simulia.com*. April 2010].

60. Phoenix Integration. *Model Center, http://www.phoenix-int.com*. April 2010].

61. PACE. *Pacelab Suite, www.pace.de*. April 2010].

62. Jarratt, T.A.W., *A model-based appraoch to support the management of engineering change*, in *St. Edmund's College*. 2004, Cambridge university.

63. Romli, F., *A strategic planning methodology for aircraft redesign*, in *Guggenheim School of Aerospace Engineering*. 2009, Georgia Institute of Technology: Atlanta.

64. Riviere, A., *Gestion de configuration et des modifications lors du développemnt de grands produits complexes en ingénieurie concourante - Cas d'application Aéronautique*, in *Génie industriel*. 2004, Institut national polytechnique de Grenoble: Grenoble.

65. Mavris, D., et al., *System Level Assessment of Uninhabited Aerial Vehicle (UAV) Utilizing Fuel Cell Technology*. 2006, Aerospace System Design Laboratory for NAVAIR through Eagle Systems, Inc.: Atlanta, Georgia

66. Soban, D. and E. Upton, *Design of a UAV to optimize use of fuel cell propulsion technology*, in *Infotech@Aerospace*. 2005: Arlington, Virginia.

67. Gavel, H., P. Krus, and J. Andersson, *Quantification of the elements in the relationship matrix, a conceptual study of aircraft fuel system*, in *42nd AIAA Aerospace Science and Sciences Meeting and Exhibit*. 2004: Reno, Nevada.

68. Gavel, H., J. Olvander, and P. Krus, *A quantified relationship matrix aided by probabilistic design and optimization*, in *26th International Congress of the Aeronautical Sciences*. 2008: Anchorage, Alaska.

69. Gavel, H., J. Olvander, and P. Krus, *Aircraft fuel system synthesis aided by interactive morphology, optimizaiton and probabistic design*, in *46nd AIAA Aerospace Science and Sciences Meeting and Exhibit*. 2008: Reno, Nevada.

70. Kirby, M.R., *Method for technology identification evaluation and selection for aircraft conceptual and preliminary design*, in *Guggenheim school of aerospace engineering*. 2001, Georgia Institute of Technology: Altanta Georgia.

71. Liscouët-Hanke, S., S. Pufe, and J.C. Maré, *A simulation framework for aircraft power management.* Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering, 2008. **222**(6): p. 749-756.

72. Liscouët-Hanke, S., *A simulation framework for aircraft power systems architecting*, in *26th Congress of International Council of the Aeronautical Sciences*. 2008: Anchorage, Alaska, USA

73. Heckmann, J., *Civil Aircraft Safet/Reliability*, in *System design S/R course*. 2001, Airbus SAS: Blagnac.

74. Department of Defense (United States of America), *Department of Defense - Interface Standard*, in *Aircraft Electric Power Characteristics*. 2004.

75. Khozikov, V., G.M. Roe, and J. Breit, *Electrical Power Generation Fuel Cell Systems for Airborne Applications and Experimental Studies on their Compliance with the Electrical Power Quality and Design Requirements and Standards*, in *7th International Energy Conversion Engineering Conference*. 2009, AIAA: Denver, Colorado.

76. Phan, L., *A Methodology for the Efficient Integration of Transient Constraints in the Design of Aircraft Dynamics Systems*, in *Guggenheim school of aerospace engineering*. 2010, Georgia Institute of Technology: Altanta Georgia.

77. Luongo, C.A., et al., *Next Generation More-Electric Aircraft: A Potential Application for HTS Superconductors.* IEEE Transaction on Applied Superconductivity, 2009. **19**(3): p. 1055-1068.

78. Masson, P.J., et al., *Superconducting Ducted Fan Design for Reduced Emissions Aeropropulsion.* IEEE Transaction on Applied Superconductivity, 2009. **19**(3): p. 1662-1668.

79. Das, I. and J.E. Dennis, *Normal-Boundary Intersection: An Alternate Method for Generating Pareto Optimal Points in Multicriteria Optimization Problems*, I. Report, Editor. 1996, NASA Contrator Report: Hampton, Virginia.

80. Felder, J.L., *Distributed Turbo-Electric Cycle Analysis*, in *Program Planning Meeting for Distributed Turboelectric Propulsion for Transport Aircraft*. 2009, NASA Glenn Research Center: Cleveland, Ohio.

81. Waters, M., *Turboelectric Regional Jet Transport Cycle Analysis & Aircraft Design* in *Program Planning Meeting for Distributed Turboelectric Propulsion for Transport Aircraft*. 2009, NASA Glenn Research Center: Cleveland, Ohio.

82. Kernstine, K., et al., *Designing the Next DC-3*, in *NASA Graduate Design Competition*. 2008, The Aerospace Systems Design Laboratory, Guggenheim School of Aerospace Engineering, Georgia Institute of Technology: Atlanta, Georgia.

83. NASA Langley Research Center, *GRC Turbo-Electric Workshop*. 2009: Ohio Aerospace Institute Auditorium.

84. Felder, J.L., H.D. Kim, and G.V. Brown, *Turboelectric Distributed Propulsion Engine Cycle Analysis for Hybrid-Wing-Body Aircraft*, in *47th AIAA Aerospace Sciences Meeting Including The New Horizons Forum and Aerospace Exposition*. 2009, AIAA: Orlando, Florida.

85. Cumpsty, N., *Jet Propulsion - A Simple Guide ti the Aerodynamic and Thermodynamic Design and Performance of Jet Engines*, ed. C.U. Press. 2005.

86. Eclipse Foundation. *http://www.eclipse.org*. April 2010].

87. Raymer, D., ed. *Aircraft Design: A Conceptual Approach*. Education Series, ed. Przemieniecki. 1992, AIAA.

88. Airbus SAS, *A320 ATA 29*, in *System Reference Note*.

89. Airbus SAS, *ATA 29*, in *Aircraft Maintenance Manual A320*.

90. Barnhart, P.J., *IPAC - Inlet Performance Analysis Code*. 1997, NASA Lewis Research Center.

91. Henry, J.R., C.C. Wood, and S.W. Wilbur, *Summary of Subsonic-Diffuser Data*. 1956, NACA Langley Aeronautical Laboratory.

92. Hill, P. and C. Peterson, *Mechanics and Thermodynamics of Propulsion*. 2nd Edition ed. 1992: Addison Wesley.

93. GE, A.E.B.G., *Energy Efficient Engine - Integrated Core/Low Spool, Design and Performance Report*. 1983, NASA Langley Research Center: Hampton Virginia.

94. NASA Glenn Research Center, *Weight Analysis of Turbine Engines (WATE) Manual*.

95. Vanderplaats, G.N., *Numerical optimization techniques for engineering design*, ed. t. edition. 2005, Colorado Springs, Colorado: Vanderplaats R&D, Inc.

96. Larminie, J. and J. Lowry, *Electric Vehicle Technology Explained*, ed. Wiley. 2003.

Index of Terms

Aircraft System Architecture, 32
Analysis model, 319
Architect, 33, 145
Architecting Team, 144
Architectural depth, 112
Architecture, 2, 83
Architecture concept, 83
Architecture structure, 97
ATA chapter, 88
Attribute, 79
Bottom-up analysis, 129
Bottom-up development, 35
Boundary function, 107
Boundary functions, 75, 233
Clustered-Concurrent Engineering, 26
Component, 2
Conceptual design phase, 16
Conceptual model, 319
Concurrent engineering, 20
Configuration, 204
Constraining requirement, 402
Criticality level, 158
Decomposition method, 84
Derivative analysis, 130
Design, 12
Design concept, 16
Design freedom, 19
Design phases, 15
Design plateau, 27
Design problem, 14
Detailed design phase, 16
Directed graphs, 99
Disciplinary-based decomposition, 85
Expert, 146
Failure configuration, 158
Figure of Merit, 79
Flight conditions, 157
Flight phase, 156
Flight phases, 236
Function, 153
Function port, 328
Functional breakdown, 233
Functional decomposition, 87
Functional description, 168
Functional flow, 172, 174
Functional profile, 156, 161
Functional requirement, 77
Functional specification, 4, 16
Functional specifications, 77
Functional tree, 73
Fundamental objective, 80
Geometric decomposition, 86
Induced functions, 75, 107
Integrated Product Team, 23
Interaction graph, 98
Interaction matrix, 99
Interactive Reconfigurable Matrix of Alternatives, 93
Knowledge, 19
Leapfrogging, 6
MDA, 146
Means objective, 80
Measure of Effectiveness, 79
Mission parameters, 156, 233
Mission sequence, 208, 219
Model components, 321

Modeling bricks, 344
Morphological matrix, 92
Mssion envelope, 156
Multi-Disciplinary Analysis, 23
Objective, 79
Operational scenarios, 208
Operational specification, 77
Operational state, 219
Opportunistic requirement, 402
Outsourcing of developments, 6
Philosophical argumentation
  Hypothesis 1, 161, 209, 223, 232, 425
  Hypothesis 2, 199, 308, 309, 393, 427
  Hypothesis 2.1, 172, 199, 223, 308, 309, 390
  Hypothesis 2.2, 216, 308, 391, 393
  Hypothesis 3.1, 180, 186, 251, 263, 264, 269, 425
  Hypothesis 3.2, 195, 264, 269, 284, 290, 303, 426
  Hypothesis 3.3, 192, 196, 264, 265, 268, 269, 303, 426
  Hypothesis 3.4, 197, 264, 269, 291, 302, 303, 426
  Objective 1, 67, 143, 229, 310
  Objective 2, 67, 229, 303, 310, 422
  Objective 3, 68, 105, 229, 251, 395
  Objective 4, 68, 105, 143, 229, 251, 263, 269, 414
  Objective 5, 68, 126, 229, 266, 269, 303, 418, 422
  Research Question 0, 67, 229
  Research Question 1, 69, 147, 152, 229
  Research Question 2, 69, 133, 148, 198, 229
  Research Question 2.1, 199, 393
  Research Question 2.2, 199, 212
  Research Question 2.2.1, 213, 214, 216, 391, 393
  Research Question 2.2.2, 213, 214, 223, 390
  Research Question 3, 69, 148, 180, 229
  Research Question 3.1, 180, 263
  Research Question 3.2, 181, 186, 264
  Research Question 4, 69, 160, 229, 264
  Research Question 4.2, 196, 303
  Research Question 5, 148, 187, 188, 196, 268
Physical decomposition, 85
Preliminary design phase, 16
Priority factor, 184, 196, 268
Production phase, 17
Propagation of Change, 130
Quality Function Deployment, 116
Requirement definition phase, 16
Scenario, 156, 177
Self interaction matrix, 98
Showstopper, 16, 21
Sizing, 31
Subsystem, 2, 32, 83
Subsystem type, 166
Symmetric, 354
Synthesis, 31
Top-down development, 35
Unified Modeling Language, 95
Voluntary functional degradation, 158

Vita

Cyril de Tenorio

Cyril de Tenorio received his Bachelors degree in aerospace engineering from Virginia Tech in 2005 and obtained his Masters degree at the Georgia Institute of Technology in summer 2007. His undergraduate research focused on turbofan and wind tunnel testing which lead him to participate in the flying test bed campaign of the T900 with Airbus as part of the A380 developments. This experience fostered his interest in aircraft power architectures. In summer 2006, he participated in a bleed/bleed-less trade-off for the A350 program. In parallel, his research focused on design methods for complex systems. Fall 2007, Cyril became the EADS fellow at the Georgia Institute of Technology. As part of this fellowship, he pursued his PhD in collaboration with Airbus future projects, and EADS Innovation Works.